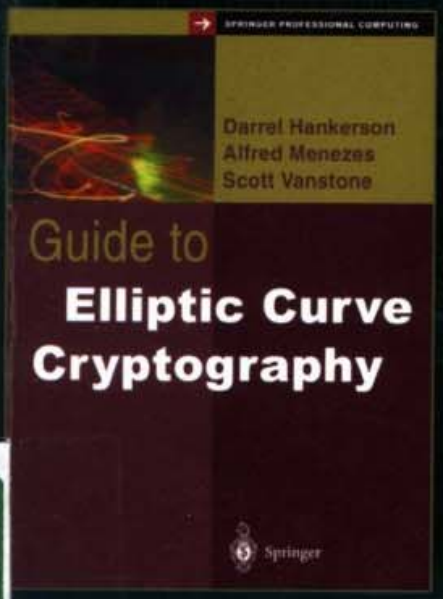
国外计算机科学教材系列

椭圆曲线密码学导论

Guide to Elliptic Curve Cryptography



Darrel Hankerson
[加] Alfred Menezes 著
Scott Vanstone

张焕国 等译

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

经典教材

椭圆曲线密码学导论

Guide to Elliptic Curve Cryptography

经过20多年的研究与开发后，椭圆曲线密码学现已得到了广泛的传播与应用。产业界、银行和政府标准的制定促使了这种有效的公钥机制的应用。通过广泛描述椭圆曲线密码学的实际应用，本书解释了基本的数学知识，描述了最新的实现方法，给出了公钥加密、数字签名和密钥建立的标准协议。此外，本书还强调了软件和硬件实现中的一些重要问题，以及旁门攻击和对策。读者在了解理论基础之后，可增强对椭圆曲线密码学的实际理解。

本书自成体系，适用于计算机科学、计算机工程、网络设计和网络数据安全领域的工程技术人员、专家以及研究人员，也可供研究生作为教材使用。本书配套网站的网址为http://www.cacr.math.uwaterloo.ca/ecc/，读者可通过该网站获得本书的更新信息以及勘误信息，以及作者的相关信息。

关键主题

- 有限域和椭圆曲线算术
- 公钥密码学
- 密码协议
- 旁门攻击及其对策
- 参数组
- ECC 标准

特 性

- 以自成体系的方法介绍了椭圆曲线密码机制
- 描述了重要的产业及政府协议，如美国国家标准与技术局制定的FIPS 186-2标准
- 提供了有效实现有限域和椭圆曲线算术的技术
- 浓缩了复杂的数学知识与算法，以便于读者理解
- 包含了关于示例参数、ECC 标准及软件工具的有用参考文献、算法以及附录

Darrel Hankerson Auburn 大学数学教授，产业顾问；同时是几本关于密码学与数据压缩的图书的合著者。

Alfred Menezes 滑铁卢大学数学教授，产业顾问；同时也是几本关于密码学的图书的合著者。

Scott Vanstone 滑铁卢大学 NSERC/Pitney Bowes 产业研究会主席；同时与 A. Menezes 和 P. van Oorschot 一道撰写了 Applied Cryptography Handbook（《应用密码学手册》，已由电子工业出版社于2005年6月出版）一书，并且是知名的 Certicom 公司的奠基人。

张焕国：武汉大学计算机科学与技术学院教授、博士生导师。主要从事信息安全、容错计算和计算机应用方面的教学与科研工作。现任中国密码学会理事，中国计算机学会容错专业委员会委员，湖北省电子学会副理事长，湖北省暨武汉市计算机学会理事。

ISBN 7-121-01602-8

9 787121 016028 >

华信教育资源网 www.hxedu.com.cn

责任编辑：谭海平

责任美编：毛惠庚

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书

ISBN 7-121-01602-8 定价：38.00 元

国外计算机科学教材系列

椭圆曲线密码学导论

Guide to Elliptic Curve Cryptography

Darrel Hankerson
[加] Alfred Menezes 著
Scott Vanstone

张焕国 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

自 1984 年 Hendrik Lenstra 给出基于椭圆曲线的整数因子分解算法后，椭圆曲线在密码学和计算机数论中的应用已得到了很大的发展。椭圆曲线密码属于公钥密码体制，它可以提供与RSA密码体制同样的功能，其安全性建立在椭圆曲线离散对数问题的困难性之上。本书主要是为安全专业人员、技术开发人员和致力于椭圆曲线密码应用的人员提供的，介绍了有限域算法，椭圆曲线算术，针对椭圆曲线离散对数问题的攻击，椭圆曲线参数组和密钥对的生成与确认，数字签名、公钥加密和密钥协商的椭圆曲线协议选择，椭圆曲线密码的软件和硬件实现，旁门攻击技术及其预防对策，等等。此外，本书还为读者提供了大量的算法与参考文献。

本书自成体系，适用于计算机科学、计算机工程、网络设计和网络数据安全领域的工程技术人员、专家以及研究人员，也可供大专院校高年级本科生和研究生作为教材使用。

Translation from the English language edition: Guide to Elliptic Curve Cryptography by Darrel Hankerson, Alfred Menezes, and Scott Vanstone.
Copyright © 2004 Springer-Verlag New York, Inc.
Springer-Verlag is a company in the BertelsmannSpringer Publishing group.
All Rights Reserved.
Authorized Simplified Chinese language edition by Publishing House of Electronics Indury. Copyright © 2005.

本书中文简体字翻译版由斯普林格出版公司授予电子工业出版社。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

版权贸易合同登记号　图字：01-2004-6032

图书在版编目（CIP）数据

椭圆曲线密码学导论/（加）汉克森（Hankerson, D.）等著；张焕国等译。-北京：电子工业出版社，2005.8
（国外计算机科学教材系列）
书名原文：Guide to Elliptic Curve Cryptography
ISBN 7-121-01602-8

Ⅰ.椭...　Ⅱ.①汉...　②张...　Ⅲ.密码-理论-教材　Ⅳ.TN918.1

中国版本图书馆CIP数据核字（2005）第085074号

责任编辑：谭海平
印　　刷：北京市顺义兴华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 980　1/16　　印张：19　　字数：425千字
印　　次：2005年8月第1次印刷
定　　价：38.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

出 版 说 明

21世纪初的5至10年是我国国民经济和社会发展的重要时期，也是信息产业快速发展的关键时期。在我国加入WTO后的今天，培养一支适应国际化竞争的一流IT人才队伍是我国高等教育的重要任务之一。信息科学和技术方面人才的优劣与多寡，是我国面对国际竞争时成败的关键因素。

当前，正值我国高等教育特别是信息科学领域的教育调整、变革的重大时期，为使我国教育体制与国际化接轨，有条件的高等院校正在为某些信息学科和技术课程使用国外优秀教材和优秀原版教材，以使我国在计算机教学上尽快赶上国际先进水平。

电子工业出版社秉承多年来引进国外优秀图书的经验，翻译出版了“国外计算机科学教材系列”丛书，这套教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求，广大师生可自由选择和自由组合使用。这些教材涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。同时，我们也适当引进了一些优秀英文原版教材，本着翻译版本和英文原版并重的原则，对重点图书既提供英文原版又提供相应的翻译版本。

在图书选题上，我们大都选择国外著名出版公司出版的高校教材，如Pearson Education培生教育出版集团、麦格劳－希尔教育出版集团、麻省理工学院出版社、剑桥大学出版社等。撰写教材的许多作者都是蜚声世界的教授、学者，如道格拉斯·科默（Douglas E. Comer）、威廉·斯托林斯（William Stallings）、哈维·戴特尔（Harvey M. Deitel）、尤利斯·布莱克（Uyless Black）等。

为确保教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本系列教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师、博士，也有积累了几十年教学经验的老教授和博士生导师。

在该系列教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括对所选教材进行全面论证；选择编辑时力求达到专业对口；对排版、印制质量进行严格把关。对于英文教材中出现的错误，我们通过与作者联络和网上下载勘误表等方式，逐一进行了修订。

此外，我们还将与国外著名出版公司合作，提供一些教材的教学支持资料，希望能为授课老师提供帮助。今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

电子工业出版社

教材出版委员会

主　任	杨芙清	北京大学教授 中国科学院院士 北京大学信息与工程学部主任 北京大学软件工程研究所所长
委　员	王　珊	中国人民大学信息学院院长、教授
	胡道元	清华大学计算机科学与技术系教授 国际信息处理联合会通信系统中国代表
	钟玉琢	清华大学计算机科学与技术系教授 中国计算机学会多媒体专业委员会主任
	谢希仁	中国人民解放军理工大学教授 全军网络技术研究中心主任、博士生导师
	尤晋元	上海交通大学计算机科学与工程系教授 上海分布计算技术中心主任
	施伯乐	上海国际数据库研究中心主任、复旦大学教授 中国计算机学会常务理事、上海市计算机学会理事长
	邹　鹏	国防科学技术大学计算机学院教授、博士生导师 教育部计算机基础课程教学指导委员会副主任委员
	张昆藏	青岛大学信息工程学院教授

译 者 序

随着计算机和网络技术的高速发展和广泛应用，社会的信息化程度越来越高，但社会对计算机和网络系统的依赖性也越来越大。若计算机和网络系统的安全受到危害，则会危及国家的安全，引起社会的混乱，从而造成重大损失。因此，确保计算机和网络系统的安全已成为世人关注的社会问题，并成为计算机科学技术的热点领域。

众所周知，确保信息系统的安全是一个系统工程，必须从整体考虑，从底层着手，综合采取措施，才能比较有效地确保信息系统的安全。硬件结构的安全和操作系统的安全是信息系统安全的基础，而密码等则是关键技术。

密码技术是一门古老的技术，大概自人类社会出现战争便产生了密码。密码的发展经历了由简单到复杂、由古典到近代的发展历程，并且在它发展的各个阶段，都为确保信息安全发挥了重要作用。

1949 年香农发表了题为《保密系统的通信理论》这一著名论文，该论文把密码置于坚实的数学基础之上，标志着密码学作为一门科学的形成。1976 年 W. Diffie 和 M. E. Hellman 提出了公钥密码的概念，从此开创了一个密码新时代。公钥密码从根本上克服了传统密码在密钥分配上的困难，特别适合计算机网络应用，而且很容易实现数字签名，因而特别受到重视。目前，公钥密码已得到了广泛应用。

在国际上研究比较充分而且公认比较安全的公钥密码有，基于大整数因子分解困难性的 RSA 密码，基于离散对数问题困难性的 ELGamal 密码，以及基于椭圆曲线离散对数问题困难性的椭圆曲线密码等。

人们对椭圆曲线的研究已有 100 多年的历史，而椭圆曲线密码是 Neal Koblitz 和 Victor Miller 于 1985 年提出来的。目前，椭圆曲线密码已成为除 RSA 密码之外呼声最高的公钥密码之一。它可以提供同 RSA 密码体制同样的功能。然而它的安全性建立在椭圆曲线离散对数问题（ECDLP）的困难性之上。现在求解 ECDLP 的最好算法具有全指数时间复杂度，与此不同，整数因子分解问题却有亚指数时间算法。这意味着要达到期望的安全程度，椭圆曲线密码可以使用较 RSA 密码更短的密钥。普遍认为 160 位椭圆曲线密码可提供相当于 1024 位 RSA 密码的安全程度。由于密钥短，所以工程实现的

加解密速度较快，并且可节省能源、带宽和存储空间。正因为如此，一些国际标准化组织已把椭圆曲线密码作为新的信息安全标准。由于椭圆曲线密码具有上述优点，因此椭圆曲线密码特别适于在航空、航天、卫星及智能卡等系统中应用。

加拿大滑铁卢大学的应用密码学研究中心（CACR）是国际著名的椭圆曲线密码研究中心之一。他们在椭圆曲线密码理论与技术方面的研究成果享誉世界。本书的三位作者都是著名的椭圆曲线密码专家。其中 Darrel Hankerson 是 Auburn 大学的研究人员，2000 年至 2004 年夏季到滑铁卢大学的应用密码学研究中心（CACR）做访问学者。Alfred Menezes 和 Scott Vanstone 都是滑铁卢大学的教授。

本书全面地介绍了椭圆曲线密码的理论、技术与应用，试图为信息安全专业技术人员和椭圆曲线密码应用人员提供一本椭圆曲线密码的基础书籍。本书的一个显著特点是关键技术都给出了算法，从而为工程实现奠定了基础。内容的叙述是面向各种读者的，以使各类读者都能得到他们所需的知识。而且在每章的末尾给出了广泛的论述，为读者提供了有用的参考。

本书的第 1 章概述了密码和椭圆曲线密码。第 2 章介绍了有限域。第 3 章介绍了椭圆曲线算术。第 4 章介绍了椭圆曲线密码的各种协议。第 5 章详细介绍了椭圆曲线密码的软件和硬件实现技术。本书内容丰富，既有理论基础，又有工程实现技术。关键技术算法化，便于实现应用。讲述深入浅出，便于理解。本书是一本难得的好书，既可用做研究生和高年级本科生的教材，也可供从事信息安全、计算机、通信、电子等领域的科技人员参考。

本书的前言、第 1 章、第 2 章和第 3 章由张焕国翻译，第 4 章、第 5 章、第 6 章和附录由王张宜翻译。孟庆树、王邦菊等帮助整理译稿。全书由张焕国统稿和审校。由于译者的专业知识和外语水平有限，书中错误在所难免，敬请读者指正，译者在此先致感谢之意。

译者于武汉珞珈山

2005.8

前　言

代数、代数几何和数论学者对椭圆曲线的研究可追溯到 19 世纪中叶。现在已有大量描述椭圆曲线美好和优雅特性的论著。1984 年，Hendrik Lenstra 给出了一个独具创意的基于椭圆曲线的整数因子分解算法。这一成功推动了研究工作者去开拓椭圆曲线在密码学和计算数论中的其他应用。

公钥密码的概念是由 W. Diffie 和 M. Hellman 于 1976 年提出的。第一个实际的公钥密码是 1977 年 R. Rivest，A. Shamir 和 L. Adleman 提出的 RSA 公钥密码体制。RSA 密码是现在最著名的公钥密码之一，它的安全性建立在整数因子分解的困难性之上。椭圆曲线密码(ECC)是 1985 年由 N. Koblitz 和 V. Miller 提出的。椭圆曲线密码属于公钥密码体制，它可以提供同 RSA 密码体制同样的功能。然而它的安全性建立在椭圆曲线离散对数问题(ECDLP)的困难性之上。现在求解 ECDLP 最好的算法具有全指数时间复杂度，与此不同，整数因子分解问题却有亚指数时间算法。这意味着对于达到期望的安全程度，椭圆曲线密码可以使用较 RSA 密码更短的密钥。例如，普遍认为 160 位的椭圆曲线密码可提供相当于 1024 位 RSA 密码的安全程度。由于密钥短而获得的优点包括加解密速度快、节省能源、节省带宽和节省存储空间。

受众　本书试图为安全专业人员、技术开发人员和致力于椭圆曲线密码应用的人员提供一本基础书籍。内容的叙述是面向各种读者的，并且假设读者具有不高于大学程度的计算机科学、工程和数学的基础。本书不是写给理论专家的，它明显缺少数学证明。本书的覆盖面相当宽，并在每章的末尾给出了大量的文献，为读者提供了有用的参考。

总体介绍　本书第 2 章集中介绍有限域算术，第 3 章集中介绍椭圆曲线算术。第 4 章首先介绍针对 ECDLP 的各种攻击方法，然后介绍椭圆曲线参数组和密钥对的生成与确认，以及数字签名、公钥加密和密钥协商的椭圆曲线协议选择。本书没有介绍针对 ECDLP 的各种攻击的数学细节，也没有论述椭圆曲线点的计数算法，因为许多数学工作者对此已经非常熟悉（以简明易读的形式阐述这些内容本身就是一种极大的挑战）。第 2 章、第 3 章和第 4 章的内容选取主要考虑了 ECC 的标准，这些标准由公认的标准化机构所

制定，如椭圆曲线数字签名算法(ECDSA)的标准 FIPS 186-2 就是由美国国家标准与技术研究所(NIST)制定的。第 5 章详细介绍了椭圆曲线密码的软件和硬件的有效实现技术，并且还介绍了“旁门攻击”技术及其预防对策。虽然第 5 章的内容是有限的，但是我们希望它能给软件开发人员和硬件设计人员提供一些有益的工程借鉴。

致谢 非常感谢曾给过我们有益的建议和帮助的人们。他们是 Mike Brown, Eric Fung, John Goyo, Rick Hite, Rob Lambert, Laurie Law, James Muir, Arash Reyhani-Masoleh, Paul Schellenberg, Adrian Tang, Edlyn Teske 和 Christof Zalka。特别感谢 Helen D'Souza，他为本书设计了精美的艺术插图。还要感谢 Cindy Hankerson 和 Sherry Shannon-Vanstone，是他们建议在书的插图中采用自然曲线的图案。最后我们感谢 Springer 出版社的编辑 Wayne Wheeler 和 Wayne Yuhasz，感谢他们所给予的不断鼓励和支持。

读者可以通过我们的网站获得本书的更新和勘误信息，以及我们的联系信息。我们的网址是 http://www.cacr.math.uwaterloo.ca/ecc/。我们非常感激读者能够发现并告诉我们书中的错误和缺陷。

Darrel R. Hankerson, Alfred J. Menezes, Scott A. Vanstone
Auburn & Waterloo
2003.6

目　录

第 1 章　概论 …… 1

1.1　密码学基础 …… 1
1.2　公钥密码 …… 5
1.3　为什么需要椭圆曲线密码 …… 13
1.4　导读 …… 18
1.5　注释和进一步参考 …… 19

第 2 章　有限域算术 …… 22

2.1　有限域介绍 …… 22
2.2　素域算术 …… 26
2.3　二进制域算术 …… 44
2.4　最佳扩域算术 …… 58
2.5　注释和进一步参考 …… 66

第 3 章　椭圆曲线算术 …… 70

3.1　椭圆曲线的概念 …… 71
3.2　点的表示与群的运算法则 …… 81
3.3　点乘 …… 90
3.4　Koblitz 曲线 …… 108
3.5　具有有效可计算自同态的曲线 …… 118
3.6　用折半进行点乘 …… 124
3.7　点乘的开销 …… 136
3.8　注释和进一步参考 …… 142

第 4 章　密码协议 …… 147

4.1　椭圆曲线离散对数问题 …… 147

4.2 参数组 ····· 164
4.3 密钥对 ····· 172
4.4 签名方案 ····· 175
4.5 公钥加密方案 ····· 180
4.6 密钥建立 ····· 184
4.7 注释和进一步参考 ····· 188

第 5 章 实现问题 ····· 196

5.1 软件实现 ····· 196
5.2 硬件实现 ····· 213
5.3 实现的安全 ····· 225
5.4 注释和进一步参考 ····· 235

附录 A 参数举例 ····· 242

附录 B ECC 标准 ····· 251

附录 C 软件工具 ····· 254

参考文献 ····· 258

索引 ····· 286

第1章 概 论

椭圆曲线具有丰富多彩的历史，数学家对它的研究已经有一百多年。人们利用椭圆曲线来解决各种不同的问题。一个例子是用来解决同余数问题，这个问题要求把表示某些直角三角形面积的正整数进行分类，而这些直角三角形的边长是有理数。另一个例子是用来证明 Fermat 大定理。Fermat 大定理叙述的是，当 n 是大于 2 的整数时，方程 $x^n + y^n = z^n$ 对于 x, y, z 无非零整数解。

1985 年，Neal Koblitz 和 Victor Miller 分别独立地提出了利用椭圆曲线设计公钥密码体制的问题。从此以后关于椭圆曲线密码安全性和有效实现的大批研究成果被发表出来。1990 年以后，椭圆曲线密码开始得到商业界的认可，公认的标准化组织制定了椭圆曲线密码协议，一些公司在他们的安全产品中也采用了这些协议。

本章的目的是要讲述公钥密码体制相对于对称密码体制的优点，特别是要详细讲述各种椭圆曲线密码。但是，这里的讲述只是介绍性的。关于椭圆曲线密码的安全性及其有效实现将在后面几章讲述。

1.1 节将讲述密码学的基本目的以及对称密码与公钥密码的本质差别。1.2 节将讲述 RSA 密码、离散对数以及椭圆曲线密码族。1.3 节将对这些密码体制进行比较，以说明椭圆曲线密码的优点。1.4 节是本书的导读。最后，1.5 节给出进一步的密码学参考文献。

1.1 密码学基础

密码学是设计和分析确保在不安全信道上的通信安全的数学技术。

1.1.1 基本的通信模型

在图 1.1 中，实体 A(Alice)和 B(Bob)在不安全的信道上进行通信。我们假设在通信中存在一个敌手 E(Eve)，他要攻击 A 和 B 之间的各种安全服务。例如，A 和 B 利用移动通信电话网络进行通信，而 E 企图偷听他们的会话。或者，设 A 是个体 $\tilde{A}$ 的一个网站浏览器，B 是在线商店 $\tilde{B}$ 的网站，个体 $\tilde{A}$ 想通过网络在网站 B 购买 $\tilde{B}$ 的产品。在这种情况下信道就是

互联网。敌手 E 企图阅读 A 和 B 的交易信息，以便得到 $\tilde{A}$ 的信用卡信息，或假冒 A 或 B 与对方进行交易。第三个例子是考虑这样一种情况，A 通过互联网发送一封电子邮件给 B。敌手 E 企图阅读这一消息，或篡改其中的某一部分，或冒充 A 给 B 发送自己的消息。最后，考虑这样一种情况，设 A 是一个智能卡，它正在认证位于银行总部的主机 B 的读卡器 $\tilde{A}$。而敌手 E 监视通信，企图获得 $\tilde{A}$ 的账号信息，或者冒充 $\tilde{A}$，以便从 $\tilde{A}$ 的账号取走资金。显然，从这些例子可以看出通信的实体不一定是人，可以是计算机、智能卡，或涉及个人、组织（如商店、银行等）利益的软件模块。

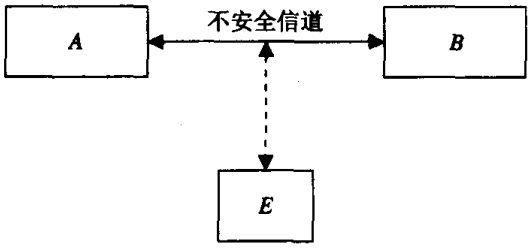


图 1.1 基本的通信模型

1.1.2 安全目标

认真考虑上述实例所揭示的保密通信的下列基本目标：

1. 秘密性：对除授权者外的任何人保持数据的秘密。即除授权者可读外，敌手 E 不可读 A 和 B 的通信消息。
2. 完整性：确保数据不被未授权者所篡改。即 B 能检测 A 发来的数据是否被 E 篡改。
3. 数据源认证：确认数据的来源。B 应能验证声称是 A 发来的数据确实是 A 发的。
4. 实体认证：确认实体的身份。即 B 应能认证通信对方的身份。
5. 不可否认性：阻止一个实体否认以前的通信或行为。即当 B 收到一个 A 发来的消息时，B 不仅能确认该消息确实为 A 所发，而且能够使公正的第三方也相信这一点。因此，A 就不能抵赖自己发给 B 的消息。

某些应用可能还有其他的安全要求，如通信实体的匿名性、访问控制（控制实体对资源的访问）等。

1.1.3 敌手模型

为了能够用模型描述 A 和 B 所面临的威胁，通常假设敌手 E 具有相当大的能力。除了能够获得信道传输的所有数据外，他还能够篡改所传输的数据并注入自己的数据。E 能够

支配相当大的计算资源。最后，除了秘密密钥以外，E 知道密码算法和通信协议的全部细节。密码工作者的挑战是面对如此强大的敌手，如何设计好的密码体制以确保通信的安全。

1.1.4 对称密码

密码体制一般可以划分为两种类型。图 1.2(a)描述了对称密码体制。对于对称密码体制，通信的双方必须就密钥的秘密性和真实性达成一致。然后，他们便可以利用对称密码，诸如数据加密标准(DES)、RC4、高级数据加密标准(AES)，来确保通信数据的秘密性。他们还可以使用消息认证码(MAC)，诸如 HMAC，确保通信数据和数据源的真实性。

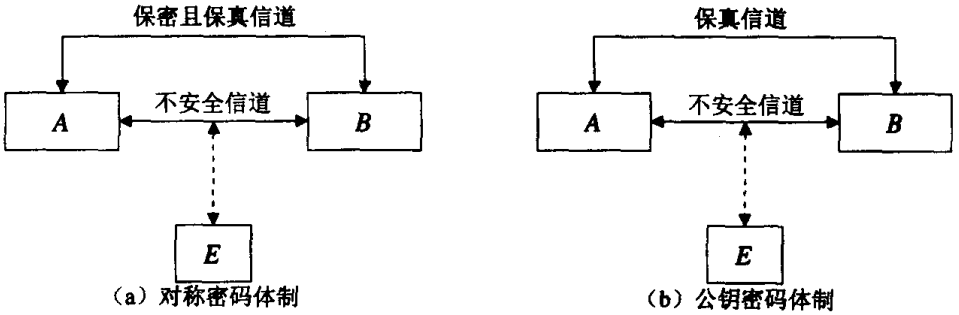


图 1.2 对称与公钥密码体制

例如，若要确保数据的秘密性，假设 A 和 B 共享密钥 k，则 A 可以用加密函数 ENC 和密钥 k 将消息 m 加密成密文 $c = \mathrm{ENC}_k(m)$，并发给 B。B 接收到 c 后用解密函数 DEC 和相同的密钥 k 解密密文 c，还原出消息 $m = \mathrm{DEC}_k(c)$。若要确保数据的完整性和数据源的真实性，则 A 和 B 首先就要协商共享一个密钥 k。然后 A 利用密钥 k 和 MAC 算法对明文消息计算认证标记 $t = \mathrm{MAC}_k(m)$。B 收到 m 和 t 后，利用 MAC 算法和同一密钥 k，对接收到的 m 重新计算认证标记 $t' = \mathrm{MAC}_k(m)$，若 $t = t'$，则认为这一消息确实是 A 所发。

密钥分发与管理 对称密码体制的最大优点是效率高，但它却存在严重的缺点。一个主要的缺点是密钥分发问题，即要求分发密钥的信道既是保密的又是保真的。在某些应用情况下，通过一个物理安全的信道，如利用可靠的信使作为安全的信道，很容易实现密钥分发。另一种方法是利用一个在线的可信第三方。开始时它为所有网络实体建立一个保密的密钥，然后利用这些保密的密钥进行通信密钥的分发[①]。这种方案特别适合具有中央权威机构的应用环境，然而这显然不适合于互联网环境下的电子邮件应用。

第二个缺点是密钥管理问题，即在有 N 个实体的网络中，每个实体都必须保存其他

① 这种方法采用一个权威的第三方分发对称密码的密钥，它已用于针对客户机/服务器系统的 Kerberos 网络认证协议中。

$N-1$ 个实体的密钥。利用一个在线的可信第三方，这一问题可以得到缓解。在需要密钥时由这一在线的可信第三方分发密钥，因此可以减小实体存储大量密钥的压力。然而，这种解决方案对有些应用情况却不适用。最后指出，因为由两个（或更多个）实体共享密钥，所以对称密码体制不能实现用于不可否认服务的数字签名，这是因为无法区分共享密钥的不同实体的行为[①]。

1.1.5 公钥密码

图 1.2(b)所示的公钥密码是 Diffie, Hellman 和 Merkle 针对对称密码体制的上述缺点，于 1975 年提出的。与对称密码不同，公钥密码仅要求密钥的交换是保真的，而不要求其是保密的。每个实体选择一个密钥对 (e,d)，其中 e 是公钥，而 d 是私钥，私钥需要保密。这种密钥具有如下特性，且仅由公钥不能计算出私钥。

秘密性 如果实体 A 希望发送一个秘密消息 m 给 B，他首先获得 B 的公钥 e_B，并利用公钥密码体制的加密函数 ENC，计算出密文 $c=\mathrm{ENC}_{e_B}(m)$。然后 A 将 c 发给 B。B 利用自己的私钥和公钥密码体制的解密函数 DEC，解密恢复出明文 $m=\mathrm{DEC}_{d_B}(c)$。这里假设敌手只知公钥 e_B 而不知私钥 d_B，因此不能解密密文 c。可以看出，其中不需要对 e_B 保密。重要的是 A 必须获得真实的 e_B；否则，若 A 得到的是冒充 B 的另一实体 E 的公钥 e_E，则 E 就可以恢复消息 m。

不可否认性 数字签名可用于数据源和数据完整性认证，提供不可否认服务。实体 A 应用数字签名产生算法 SIGN 和自己的私钥 d_A，计算产生消息的签名：$s=\mathrm{SIGN}_{d_A}(m)$。实体 B 接收到 s 和 m 后，利用 A 的真实公钥 e_A 和数字签名验证算法来验证 s 是否是从 m 和 d_A 产生的签名。因为假定只有 A 知道 d_A，所以 B 可以断定消息 m 确实是 A 所发。此外，因为签名验证只需要公开的 m 和 e_A，因此 m 的签名 s 也可由第三方验证。若 A 企图抵赖自己对消息 m 的签名，则第三方可以仲裁。与手写签名不一样，A 的签名 s 与所签的消息 m 相关，因此可以阻止简单地将 s 附加到其他消息 m' 之后冒充 A 对 m' 签名的伪造行为。虽然公钥 e_A 不需要保密，但是验证 A 的签名时需要使用 A 的真实公钥 e_A。

这样，公钥密码圆满地解决了对称密码面临的三个难题：密钥分发、密钥管理和提供不可否认性。应当指出，虽然公钥密码不需要使用保密信道分发公钥，但是用于分发和管理公钥的公钥基础设施(PKI)的实施在实践上可能会遇到极大的挑战。另外，公钥密码的运

① 也可以使用对称密码设计数字签名方案，但是这种方案不实用，因为这种方案需要使用在线的可信第三方，而且还必须在每次签名时都要换用新的密钥。

算通常比对称密码慢得多。因此，通常采用对称密码和公钥密码的混合系统，将对称密码的高效性和公钥密码的功能方便性结合起来。

下一节将介绍三种著名的公钥密码体制。

1.2 公钥密码

在公钥密码体制中，密钥对的选择要保证从公钥求出私钥等价于要求解一个困难的计算问题。构成常用公钥密码基础的困难问题有如下的数论问题：

1. 整数的因子分解问题，其困难性是 RSA 公钥密码安全性的基础。
2. 离散对数问题，其困难性是 ElGamal 公钥密码及其变体（如数字签名算法）安全性的基础。
3. 椭圆曲线离散对数问题，其困难性是椭圆曲线公钥密码安全性的基础。

本节介绍 RSA 公钥密码、ElGamal 公钥密码和椭圆曲线公钥密码的基本方案。我们强调指出，本节的介绍是教科书式的，而对于实际应用，要能抵抗各种实际攻击，还需要对这些方案进行加强，例如在加密之前用随机数进行明文填充。然而，这些基本方案描述了 RSA、离散对数和椭圆曲线公钥密码的基本思想。第 4 章将介绍椭圆曲线密码的更深入内容。

1.2.1 RSA 体制

RSA 密码以发明者 Rivest，Shamir 和 Adleman 的名字的首字母命名，他们在公钥密码的概念提出之后不久，于 1977 年提出了这种公钥密码。

RSA 密钥生成

RSA 密钥对用算法 1.1 生成。其公钥由一对整数 (n,e) 组成，其中 n 是 RSA 的运算模，它是两个同样长度的保密随机素数 p 和 q 的乘积。加密指数 e 是满足 $1<e<\phi$ 且 $\gcd(e,\phi)=1$ 的整数，其中 $\phi=(p-1)(q-1)$。私钥 d 也称为解密指数。它是满足 $1<d<\phi$ 且 $ed\equiv 1 \pmod{\phi}$ 的整数。已经证明，从公钥对 (n,e) 计算出私钥 d 等价于从 n 分解出因子 p 和 q。这一问题被称为整数的因子分解问题(IFP)。

算法 1.1 RSA 密钥对生成

输入：安全参数 l。

输出：RSA 公钥 (n,e) 和私钥 d。

1. 随机选择两个素数 p 和 q，p 和 q 的长度同为 $l/2$。
2. 计算 $n = pq$ 和 $\phi = (p-1)(q-1)$。
3. 任意选择整数 e，e 满足 $1<e<\phi$ 且 $\gcd(e,\phi)=1$。
4. 计算整数 d，d 满足 $1<d<\phi$ 且 $ed \equiv 1 \pmod{\phi}$。
5. 返回(n, e, d)。

RSA 加密方案

RSA 加密和签名方案基于以下理论依据，即对于所有的整数 m，都有

$$m^{ed} \equiv m \pmod{n}。 \tag{1.1}$$

RSA 公钥密码的基本加密和解密过程分别如算法 1.2 和算法 1.3 所示。解密的原理是根据式(1.1)导出的 $c^d \equiv (m^e)^d \equiv m \pmod{n}$。安全性建立在从密文 $c = m^e \bmod n$ 和公钥(e, n)计算明文 m 的困难性之上。这个问题就是求 c 的 mod n 的 e 次根，被认为与整数分解一样困难（尚未证明）。

算法 1.2 基本 RSA 加密

输入：RSA 公钥(n, e)，明文 $m \in [0, n-1]$。

输出：密文 c。

1. 计算 $c = m^e \bmod n$。
2. 返回(c)。

算法 1.3 基本 RSA 解密

输入：RSA 公钥(n, e)，RSA 私钥 d，密文 c。

输出：明文 m。

1. 计算 $m = c^d \bmod n$。
2. 返回(m)。

RSA 签名方案

RSA 公钥密码的签名和签名验证过程分别如算法 1.4 和算法 1.5 所示。签名者首先利用密码学杂凑函数 H，计算消息 m 的摘要 $h = H(m)$，摘要值 h 称为消息 m 的短指纹。然后，签名者用自己的私钥 d 计算 h mod n 的 e 次根 s，即 $s = h^d$ mon n。注意，根据式(1.1)有 $s^e \equiv h \pmod{n}$。最后，签名者把消息 m 和自己的签名 s 一起发给验证方。验证方根据收

到的消息重新计算消息的摘要 $h = H(m)$，并从 s 恢复出摘要 $h' = s^e \bmod n$。若 $h = h'$，则验证方认为签名 s 是真实的。签名方案的安全性建立在伪造者在不知道私钥 d 的情况下计算 $h \bmod n$ 的 e 次根的困难性之上。

算法 1.4 基本 RSA 签名生成

输入：RSA 公钥(n, e)，RSA 私钥 d，消息 m。

输出：签名 s。

1. 计算 $h = H(m)$，其中 H 是一个杂凑函数。
2. 计算 $s = h^d \bmod n$。
3. 返回(s)。

算法 1.5 基本 RSA 签名验证

输入：RSA 公钥(n, e)，消息 m，签名 s。

输出：接受或拒绝签名。

1. 计算 $h = H(m)$。
2. 计算 $h' = s^e \bmod n$。
3. 若 $h = h'$，则返回(“接受签名”)；否则，返回 (“拒绝签名”)。

RSA 运算中最费时的运算是模幂运算，如在加密中的 $m^e \bmod n$ 和解密中的 $c^d \bmod n$。为了提高加密和签名验证的效率，可以选择小的加密指数 e，在实际中常采用 $e = 3$ 和 $e = 2^{16} + 1$。解密指数 d 与 n 一样长。因此，采用小加密指数的 RSA 加密和签名验证就比 RSA 解密和签名生成快得多。

1.2.2 离散对数体制

第一个离散对数(DL)体制是 Diffie 和 Hellman 于 1976 年提出的密钥协商协议。1984 年，ElGamal 提出了离散对数公钥加密方案和离散对数公钥签名方案。此后，人们相继提出了离散对数公钥密码的各种变体。下面我们将介绍基本 ElGamal 公钥加密方案和数字签名算法(DSA)。

DL 密钥生成

在离散对数体制中，密钥对是与公开参数组 (p, q, g)联系在一起的。其中 p 是素数，q 是 $p-1$的素因子，$g \in [1, p-1]$，g 的阶为 q，即 $t = q$ 是满足 $g^t \equiv 1 \pmod{p}$ 的最小正整数。私钥 x 是从区间 $[1, q-1]$ 内随机选择的一个整数（这一操作用符号 $x \in_R [1, q-1]$ 表

示)。而相应的公钥是$y=g^x \bmod p$。对于给定的参数组(p,q,g)和y，确定x是离散对数问题(DLP)。算法 1.6 和算法 1.7 分别给出了 DL 参数组的生成和密钥对的生成过程。

算法 1.6 DL 参数组的生成

输入：安全参数l，t。

输出：DL 参数组(p,q,g)。

1. 选择一个t位的素数q和l位的素数p，使得q整除$p-1$。
2. 选择一个q阶元素g：
 2.1 任意选择一个$h \in [1,p-1]$，并计算$g=h^{(p-1)/q} \bmod p$。
 2.2 若$g=1$，则转到步骤 2.1。
3. 返回(p,q,g)。

算法 1.7 DL 密钥对的生成

输入：DL 参数组(p,q,g)。

输出：公钥y和私钥x。

1. 选择$x \in_R [1,q-1]$。
2. 计算$y=g^x \bmod p$。
3. 返回(y,x)。

DL 加密方案

基本 ElGamal 公钥密码的加密和解密过程分别如算法 1.8 和算法 1.9 所示。若y是接收者的公钥，则消息m的加密通过把m乘以$y^k \bmod p$来完成，其中k是发送者随机选择的整数。发送者将$c_2 = my^k \bmod p$和$c_1 = g^k \bmod p$发送给接收者。接收者利用自己的私钥x计算

$$c_1^x \equiv g^{kx} \equiv y^k \pmod{p},$$

再用此值去除c_2便可恢复出明文m。攻击者要恢复出明文m，必须计算$y^k \bmod p$。根据参数组(p,q,g)，y和$c_1=g^k \bmod p$计算$y^k \bmod p$被称为 Diffie-Hellman 问题(DHP)。DHP 被认为与离散对数问题一样困难(某些情况下已经证明)。

算法 1.8 基本 ElGamal 加密

输入：DL 参数组(p,q,g)，公钥y，明文$m \in [0,p-1]$。

输出：密文(c_1, c_2)。

1. 选择$k \in_R [1, q-1]$。
2. 计算$c_1 = g^k \bmod p$。
3. 计算$c_2 = m \cdot y^k \bmod p$。
4. 返回(c_1, c_2)。

算法 1.9 基本 ElGamal 解密

输入：DL 参数组(p, q, g)，私钥x，密文(c_1, c_2)。

输出：明文m。

1. 计算$m = c_2 \cdot c_1^{-x} \bmod p$。
2. 返回(m)。

DL 签名方案

数字签名算法(DSA)由美国国家标准技术研究所(NIST)于 1991 年提出，并且以美国政府的联邦信息处理标准(FIPS 186)的形式发表，命名为数字签名标准(DSS)。算法 1.10 和算法 1.11 分别给出了其签名和签名验证的过程。

实体 A 用私钥 x 签名一个消息，首先要选择一个随机数$k \in [1, q-1]$，然后计算$T = g^k \bmod p$，$r = T \bmod q$和

$$s = k^{-1}(h + xr) \bmod q, \tag{1.2}$$

其中$h = H(m)$是消息摘要。A 对消息 m 的签名是(r, s)。为了验证签名，验证实体必须检验(r, s)是否满足式(1.2)。因为验证者既不知道 A 的私钥 x，又不知道 k，所以不能直接验证式(1.2)。但是，式(1.2)等价于

$$k \equiv s^{-1}(h + xr) \pmod q。 \tag{1.3}$$

分别以式(1.3)的两边作为 g 的幂指数，得到

$$T \equiv g^{hs^{-1}} y^{rs^{-1}} \pmod p。$$

所以验证者能够计算 T，并因而能够验证$r = T \bmod q$。

算法 1.10 DSA 签名生成

输入：DL 参数组(p, q, g)，私钥x，消息m。

输出：签名(r, s)。

1. 选择$k \in_R [1, q-1]$。
2. 计算$T = g^k \mod p$。
3. 计算$r = T \mod q$。若$r = 0$，则转到步骤 1。
4. 计算$h = H(m)$。
5. 计算$s = k^{-1}(h + xr) \mod q$。若$s = 0$，则转到步骤 1。
6. 返回(r, s)。

算法 1.11 DSA 签名验证

输入：DL 参数组(p, q, g)，公钥y，消息m，签名(r, s)。

输出：接受或拒绝签名。

1. 检验r和s是否都是区间$[1, q-1]$内的整数。若有任何一个不是，则返回（“拒绝签名”）。
2. 计算$h = H(m)$。
3. 计算$w = s^{-1} \mod q$。
4. 计算$u_1 = hw \mod q$和$u_2 = rw \mod q$。
5. 计算$T = g^{u_1} y^{u_2} \mod p$。
6. 计算$r' = T \mod q$。
7. 若$r = r'$，则返回（“接收签名”）；否则，返回（“拒绝签名”）。

1.2.3 椭圆曲线体制

1.2.2 节描述的离散对数体制也可以抽象到一个有限循环群上。我们首先介绍一些群的基本概念，并讲述这种推广。然后考察椭圆曲线群，并且讲明这个椭圆曲线群是如何构成离散对数体制的。

群

一个交换群$(G, *)$由集合G和一个二元运算$*$：$G \times G \to G$构成，且满足下列性质：

(i) *结合律*：对于所有的$a, b, c \in G$，有$a*(b*c) = (a*b)*c$。

(ii) *存在单位元*：存在一个元素$e \in G$，使得$a*e = e*a = a$，对于所有的$a \in G$。

(iii) *存在逆元素*：对于每一个$a \in G$，都存在一个元素$b \in G$，使得$a*b = b*a = e$，称元素b为a的逆元素。

(iv) *交换律*：对于所有的$a, b \in G$，有$a*b = b*a$。

群的运算通常称为加法(+)或乘法(·)，并分别称为加法群和乘法群。定义了加法运算的群，称为加法群。加法群的单位元通常用 0 来表示，加法群的元素 a 的逆元素通常用 $-a$ 来表示。定义了乘法运算的群，称为乘法群。乘法群的单位元通常用 1 来表示，乘法群的元素 a 的逆元素通常用 a^{-1} 来表示。若 G 是有限集合，则称其为有限群，并称 G 中元素的个数为群的阶。

例如，令 p 是一个素数，并且 $\mathbb{F}_p=\{0,1,2,\ldots,p-1\}$ 表示模 p 的整数集合，则 $(\mathbb{F}_p,+)$ 是一个模 p 的有限加法群。其中 + 表示模 p 的整数加法，加法单位元用 0 表示。此外，用 $\mathbb{F}_p^*$ 表示 $\mathbb{F}_p$ 的全体非零元素构成的集合，运算 · 表示模 p 的整数乘法，则 $(\mathbb{F}_p^*,\cdot)$ 为 $p-1$ 阶有限乘法群，其乘法单位元是 1。进而，$(\mathbb{F}_p,+,\cdot)$ 构成有限域，并简单地记为 $\mathbb{F}_p$（见 2.1 节）。

设 G 是一个 n 阶有限乘法群，$g\in G$，称使 $g^t=1$ 的最小正整数 t 为 g 的阶。阶 t 总是存在的，而且是 n 的因子。g 的所有幂的集合 $\langle g\rangle=\{g^i:0\leqslant i\leqslant t-1\}$，对于 G 的运算也构成一个群，并称为由 g 生成的循环子群。对于加法群，也有同样的性质。设 G 是加法群，$g\in G$，使 $tg=0$ 的 n 的最小正整数因子 t 称为 g 的阶，并且由 g 生成的循环子群为 $\langle g\rangle=\{ig:0\leqslant i\leqslant t-1\}$，其中 tg 表示 t 个 g 相加。若 G 有一个 n 阶元素 g，则 G 是循环群，g 是 G 的生成元。

例如，对于 1.2.2 节定义的 DL 参数组 (p,q,g)，$(\mathbb{F}_p^*,\cdot)$ 为 $p-1$ 阶循环群，而且 $\langle g\rangle$ 是 q 阶循环子群。

扩展离散对数问题

设 $(G,\cdot)$ 是一个 n 阶循环群，g 为生成元。则我们可以表明 1.2.2 节给出的离散对数体制是建立在群 G 之上的。对于参数 g 和 n，私钥 x 是从区间 $[1,n-1]$ 内随机选择的整数，公钥 $y=g^x$。从给定的 g，n 和 y 确定出 x 的问题称为群 G 上的离散对数问题。

为了使建立在群 G 上的离散对数体制是有效的，应当有群运算的快速算法。为了安全，群 G 上的离散对数问题应当是困难的。

两个相同阶的循环群本质上是相同的，即具有相同的结构，即使它们的元素符号不同。群元素的不同表示，会使群运算的计算速度不同以及求解离散对数问题的速度不同。

实现离散对数体制的最常用的群是有限域（1.2.2 节已讨论）的乘法群的循环子群和下面将要讲到的椭圆曲线群的循环子群。

椭圆曲线群

令 p 是一个素数，$\mathbb{F}_p$ 是模 p 的有限域。$\mathbb{F}_p$ 上的椭圆曲线 E 定义为

$$y^2 = x^3 + ax + b, \tag{1.4}$$

其中 $a, b \in \mathbb{F}_p$，且满足 $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$。设 $x, y \in \mathbb{F}_p$，若数对 (x, y) 满足式(1.4)，则称 (x, y) 为曲线 E 上的一个点。定义 ∞ 为曲线 E 的无穷远点。用 $E(\mathbb{F}_p)$ 表示曲线 E 的所有点的集合。例如，若 E 是 $\mathbb{F}_7$ 上的椭圆曲线

$$y^2 = x^3 + 2x + 4,$$

则 E 上的点是

$$E(\mathbb{F}_7) = \{\infty, (0,2), (0,5), (1,0), (2,3), (2,4), (3,3), (3,4), (6,1), (6,6)\}。$$

有一种把两个椭圆曲线点相加得到第三个点的著名方法（见 3.1 节）。加法规则采用 $\mathbb{F}_p$ 上的一些算术运算（加、减、乘、逆）对点的坐标 x_1，y_1，x_2，y_2 进行运算。对于这种加法运算，$E(\mathbb{F}_p)$ 构成一个加法交换群，并且 ∞ 为其无穷远点。群 $E(\mathbb{F}_p)$ 的循环子群可以用来构建离散对数体制。

我们将在后面通过用椭圆曲线模拟实现 1.2.2 节所介绍的离散对数体制的方法，讲述椭圆曲线密码体制的基本思想。我们还将在第 4 章广泛地讲述这样的椭圆曲线密码体制和用椭圆曲线模拟实现 DSA 签名方案的方法。

椭圆曲线密钥生成

令 E 是 $\mathbb{F}_p$ 上的椭圆曲线，P 是 $E(\mathbb{F}_p)$ 上的点，设 P 的阶是素数 n，则集合

$$\langle P \rangle = \{\infty, P, 2P, 3P, \ldots, (n-1)P\}$$

是由 P 生成的椭圆曲线循环子群。素数 p，椭圆曲线方程 E，点 P 和阶 n 构成公开参数组。私钥是在区间 $[1, n-1]$ 内随机选择的正整数 d，相应的公钥是 $Q = dP$。由公开参数组和公钥 Q 求私钥 d 的问题是椭圆曲线离散对数问题(ECDLP)。

算法 1.12 椭圆曲线密钥对的生成

输入：椭圆曲线参数组 (p, E, P, n)。

输出：公钥 Q 和私钥 d。

1. 选择 $d \in_R [1, n-1]$。
2. 计算 $Q = dP$。
3. 返回 (Q, d)。

椭圆曲线加密方案

用椭圆曲线模拟实现基本 ElGamal 方案的加密和解密过程分别由算法 1.13 和算法 1.14 给出。首先把明文 m 表示成一个点 M，然后再加上 kQ 进行加密，其中 k 是随机选择的正整数，Q 是接收者的公开密钥。发方把密文 $C_1 = kP$ 和 $C_2 = M + kQ$ 发给接收方。接收方用自己的私钥 d 计算

$$dC_1 = d(kP) = k(dP) = kQ,$$

进而可恢复出明文 $M = C_2 - kQ$。攻击者若想要恢复出明文 M，则需要计算 kQ。而从公开的参数组，Q 和 $C_1 = kP$ 计算 kQ 则是椭圆曲线的 Diffie-Hellman 问题。

算法 1.13 基本椭圆曲线加密

输入：椭圆曲线参数组 (p, E, P, n)，公钥 Q，明文 m。

输出：密文 (C_1, C_2)。

1. 将明文 m 表示为 $E(\mathbb{F}_p)$ 上的点 M。
2. 选择 $k \in_R [1, n-1]$。
3. 计算 $C_1 = kP$。
4. 计算 $C_2 = M + kQ$。
5. 返回 (C_1, C_2)。

算法 1.14 基本椭圆曲线解密

输入：椭圆曲线参数组 (p, E, P, n)，私钥 d，密文 (C_1, C_2)。

输出：明文 m。

1. 计算 $M = C_2 - dC_1$，并从点 M 取出明文 m。
2. 返回 (m)。

1.3 为什么需要椭圆曲线密码

在为某一应用选择一个公钥密码体制时，必须考虑一些准则。这些准则如下所示：

1. 功能性。该公钥密码体制能提供所需的功能吗？
2. 安全性。该公钥密码体制是安全的吗？
3. 性能。对于所希望的安全性，该公钥密码体制能满足性能要求吗？

影响抉择的其他因素还有，是否由标准化组织制定了实用的标准？是否有商业化的密码产品？专利保护？现有的开发应用程度？等等。

1.2 节介绍的 RSA，DL 和 EC 都能提供公钥密码的基本功能：加密、签名和密钥协商。几年之后，研究者发展了应用 RSA，DL 和 EC 的设计技术，并且在合理的假设下证明了它们的安全性。剩下的基本安全问题是这些密码所依赖的数学问题的困难性必须是足够的，即对于 RSA 体制的整数因子分解问题，对于 DL 体制的离散对数问题，对于 EC 体制的椭圆曲线离散对数问题。这些问题的困难性直接影响着这些密码体制的性能，因为它决定着这些密码的密钥和参数组的大小。这反过来又影响到所依赖的算术运算的性能。

在本节最后，我们概述一下整数因子分解算法、离散对数求解算法和椭圆曲线离散对数求解算法的发展状况。我们给出 RSA，DL 和 EC 密码体制获得相同安全性时参数大小的比较。比较表明椭圆曲线密码特别适合那些需要高安全性的应用。

我们首先引入一些算法分析的相关概念。

算法有效性的度量

算法的有效性用所消耗的资源来度量。最常用的度量是算法所使用的时间，有时还用算法所使用的空间或 CPU 的数目作为度量。显然，对于较大的算法输入，算法将消耗较多的资源。因此算法的有效性可以看成是算法输入规模的函数。这里，算法输入的规模定义为用合适的编码表示算法的输入所需要的位数。例如，一种对整数 n 进行因子分解的算法的输入规模为 $l=\lfloor \mathrm{lb}\, n \rfloor+1$ 位。

最有用的是算法运行时间的表达式，如果这一表达式与算法所运行的具体平台无关。这可以通过计量算法执行的基本运算（如位操作）的数量来实现。算法在最坏情况下的执行时间是一个上界，并把它表示成算法输入规模的函数，它是算法执行的基本步数的函数。例如，通过试除所有小于等于 $\sqrt{n}$ 的因子的方法来进行整数 n 的因子分解，则算法运行时间可近似地表示为 $\sqrt{n} \approx 2^{l/2}$ 次除法运算。

通常，要找出算法运行时间的精确表达式是困难的。在这种情况下，通常采用“大 O”表示法。设 f 和 g 是两个定义在正整数上的正实函数，若存在正常数 c 和 L，对于所有 $l \geqslant L$，满足 $f(l) \leqslant cg(l)$，则我们记为 $f = O(g)$。非正式地说，这意味着从渐近性看，$f(l)$ 的增长不快于一个常数与 $g(l)$ 的乘积。同样，也可以采用“小o”表示法。若对任意的正整数 c 都存在一个常数 L，对于所有 $l \geqslant L$，满足 $f(l) \leqslant cg(l)$，则记为 $f = o(g)$。非正式地说，这意味着对于大的 l，相对于 $g(l)$，$f(l)$ 变得小了。

通常，可接受的算法有效性是指算法的运行时间以输入规模的某一个多项式为界。

定义 1.15 设 A 是一个算法，其输入的长度为 l。

(i) 若对于某一正整数 $c>0$，算法的运行时间是 $O(l^c)$，则称 A 是多项式时间算法。

(ii) 若对于任意正整数 $c>0$，算法的运行时间都不具有 $O(l^c)$形式，则称 A 是指数时间算法。

(iii) 若 A 不是多项式时间算法，且算法的运行时间是 $O(2^{o(l)})$，则称 A 是亚指数时间算法。

(iv) 若算法的运行时间都不具有 $O(2^{o(l)})$形式，则称 A 是全指数时间算法。

应当指出，一个亚指数时间算法也是一个指数时间算法。要特别强调的是，它不是多项式时间算法。然而，亚指数时间算法的运行时间比全指数时间算法增长得慢。整数因子分解和离散对数求解问题都存在亚指数算法。

例 1.16（亚指数时间算法）设 A 是一个算法，其输入是整数 n 或是模 n 的小整数集合，（输入规模为 $O(\mathrm{lb}\, n)$）。若 A 的运行时间具有形式

$$L_n[\alpha, c] = O(e^{(c+o(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}}),$$

其中 c 是一个正常数，α 是满足 $0<\alpha<1$ 的一个常数，则 A 是一个亚指数时间算法。可以看出，若 $\alpha=0$，则 $L_n[0,c]$ 是 $\mathrm{lb}\, n$ 的一个多项式表达式，因此 A 是一个多项式时间算法。而若 $\alpha=1$，则 $L_n[1,c]$ 是 $\mathrm{lb}\, n$ 的一个全指数表达式，所以 A 是全指数时间算法。因此参数 α 是一个把亚指数时间算法变成多项式时间算法或全指数时间算法的很好的基准。

求解整数因子分解和离散对数问题

下面，我们简单介绍一下整数因子分解算法、离散对数求解算法和椭圆曲线离散对数求解算法的发展状况。

整数因子分解算法 设要分解的整数 n 是两个 $l/2$ 位的素数的乘积，则整数因子分解算法的输入规模是 $O(l)$ 位。已知的分解这样的整数 n 的最快算法是数域筛法(NFS)，它具有亚指数的期望运行时间

$$L_n[\frac{1}{3}, 1.923]。 \tag{1.5}$$

数域筛法(NFS)分为两个阶段：筛阶段（一些确定的关系集成在一起）和矩阵阶段（求解大规模稀疏线性方程组）。筛阶段易于并行，能够通过互联网在工作站集群上运行。然而，为了提高筛法的效率，要求每个工作站应有大容量的主存。而矩阵阶段不易并行，因为每个

处理器需要频繁地与其他处理器通信。因此它工作在一台巨型并行机上比工作在松耦合的网络工作站上更有效。

根据 2003 年的报道，用数域筛法(NFS)分解的最大 RSA 模是 530 位二进制数（160 位十进制数）。

求解离散对数问题的算法 首先回忆一下具有参数 p 和 q 的离散对数问题，其中 p 是一个 l 位长的素数，q 是一个 t 位长的素数，且 q 是 $p-1$ 的因子。因此问题的输入规模为 $O(l)$ 位。已知的求解离散对数问题的最快算法有数域筛法(NFS)，它具有亚指数的期望运行时间

$$L_p[\frac{1}{3}, 1.923]; \tag{1.6}$$

以及 Pollard's rho 算法，它的期望运行时间为

$$\sqrt{\frac{\pi q}{2}}。 \tag{1.7}$$

上面关于整数因子分解 NFS 的论述也适于求解离散对数的 NFS。Pollard's rho 算法易于并行，因此单个处理器间不需要彼此通信，只需要与一个中央处理器进行相应的通信。此外，该算法本身的存储很小，而且只需要很小的主存。

选择求解给定 DLP 的方法取决于参数 p 和 q 的规模，这本身也确定了式(1.6)和式(1.7)哪个表示的时间开销更小。实际上，DL 参数的选择使得式(1.6)和式(1.7)的期望运行时间近似相等。

根据 2003 年的报道，用数域筛法(NFS)解决的最大 DLP 实例是 397 位素数 p（120 位十进制数）。

求解椭圆曲线离散对数问题的算法 首先回忆一下椭圆曲线离散对数问题(ECDLP)。所谓 ECDLP，是要寻找一个整数 $d \in [1, n-1]$，使得 $Q = dP$，其中 n 是 t 位长的素数，P 是 $\mathbb{F}_p$ 上的椭圆曲线的一个 n 阶点，并且 $Q \in \langle P \rangle$。若我们假设 $n \approx p$，在实际应用中通常如此，则输入的规模是 $O(t)$ 位。求解 ECDLP 的已知最好算法是 Pollard's rho 算法（见 4.1 节），它的期望运行时间是

$$\frac{\sqrt{\pi n}}{2}。 \tag{1.8}$$

上面关于求解普通离散对数问题(DLP)的 Pollard's rho 算法的论述，也适于求解椭圆曲线离散对数问题(ECDLP)。

根据 2003 年的资料，用 Pollard's rho 算法求解的最大 ECDLP 实例是 109 位素域上的一个椭圆曲线离散对数问题。

密钥规模的比较

假设上面提到的整数分解算法、离散对数求解算法、椭圆曲线离散对数求解算法是最好的算法，则使 RSA、DL 和 EC 密码体制能够提供可比较的安全程度所需的参数规模已经评估给出（见表 1.1）。然而，我们无法预计将来在设计更快的算法或研制出巨型量子计算机方面可能出现的重大突破[①]。

若执行时间是算法有效性的惟一度量，则 RSA，DL 和 EC 密码体制提供等价安全性的参数规模就能够根据式(1.5)、式(1.6)、式(1.7)和式(1.8)求出。参数的规模也称为密钥的规模或密钥的长度。能够提供与 80，112，128，192 和 256 位的对称密码等价安全性的 RSA、DL 和 EC 密码的参数规模示于表 1.1 中。若安全性的级别是 k 位，则意味着用已知的最好算法进行攻击，大约需要计算 2^k 步。表 1.1 中所列出的 5 个具体的安全性级别，分别表示穷举 SKIPJACK、Triple-DES、128 位 AES、192 位 AES 和 256 位 AES 的一个密钥所需的计算工作量。

表 1.1 RSA、DL 和 EC 提供等效安全级别的密钥长度。分别给出了 DL 的参数 q 和 EC 的参数 n，RSA 的模 n 和 DL 的模 p，要达到等效安全级别应采用的密钥的长度

	安全级别（位）				
	80 （SKIPJACK）	112 （3DES）	128 （小规模 AES）	192 （中规模 AES）	256 （大规模 AES）
DL 的参数 q EC 的参数 n	160	224	256	384	512
RSA 的模 n DL 的模 p	1024	2048	3072	8192	15 360

稍微有点不能令人满意的是在表 1.1 中，密钥规模的比较仅根据数域筛法(NFS)和 Pollard's rho 算法所需要的时间进行比较。实际上，有一些限制 NFS 执行效率的因素。如筛态所需的存储空间的大小以及矩阵态下矩阵的大小和难于并行化等因素。而这些因素对于 Pollard's rho 算法的分析却不存在。给出全面考虑运行时间和软硬件等各种因素的总成本的等价密钥规模是可能的。然而要精确估计这些成本是困难的。况且最近的研究表明，通

① 人们已经知道利用量子计算机进行整数因子分解、求解离散对数问题和求解椭圆曲线离散对数问题的有效算法（参见 4.7 节中的注释）。然而，人们尚不知道大型量子计算机是否能实际制造出来。

过构建专用硬件设备可使筛态和矩阵态的处理过程的总成本大大下降。采用保守的方法并且只用时间作为 NFS 和 Pollard's rho 算法有效性的度量，看来是谨慎的。

表 1.1 的比较说明，对于给定的安全级别，ECC 比 RSA 和 DL 有更小的参数。对于更高的安全性级别，参数大小的差异更明显。更小的参数带来的好处是，计算速度更快、密钥更短和密钥证书更小。实际上，ECC 的私钥运算（如签名和解密）比 RSA 和 DL 的私钥运算要快许多倍。ECC 的公钥运算（如签名验证和加密）比 DL 的公钥运算要快许多倍。若 RSA 选择了一个小的加密指数 e（如 $e=3$ 或 $e=2^{16}+1$），则 RSA 的公钥运算可以比 ECC 稍微快一点。在处理能力、存储空间、带宽、功耗等受限的环境中，ECC 的这些优点将是很重要的。

1.4 导读

在实现一个椭圆曲线密码之前，要首先针对有限域、椭圆曲线和密码协议进行如下选择：

1. 一个有限域，一种有限域的表示方法，实现有限域算术运算的算法。
2. 一条椭圆曲线，一种椭圆曲线点的表示方法，实现椭圆曲线算术运算的算法。
3. 一个协议和实现协议算术运算的算法。

有许多影响选择的因素。为了能对具体应用提供一个最佳的解决方案，应当同时考虑所有这些因素。这些因素包括安全性、应用平台（软件和硬件）、具体计算环境的限制（如处理速度、代码空间(ROM)、存储空间(RAM)、门的数量、功耗）、具体通信环境的限制（如带宽、响应时间）等。

不必惊讶，即使不是不可能的，要选择一个最佳的全面解决方案也是十分困难的。例如，对于工作站应用的最佳方案就与智能卡应用的最佳方案有很大的不同。本书的目的就是要给信息安全实际工作者提供大量的椭圆曲线密码的各种实现和安全考虑，以便对于实际应用能做出最佳选择。

本书的其余部分的安排如下。第 2 章将给出有限域的简单介绍，并对素域、二进制域和最佳扩域，给出其最适合软件实现的算术运算算法。

第 3 章将简单介绍椭圆曲线密码，给出椭圆曲线点的表示和实现椭圆曲线算术运算的各种方法。此外，还将介绍加速 Koblitz 曲线算术运算的技术和具有有效可计算自同态的其他椭圆曲线。

第 4 章将介绍椭圆曲线密码协议，包括数字签名协议、公钥加密协议、密钥传输协议。

此外，还将研究椭圆曲线密码工作参数和密钥对的生成和检验，以及求解椭圆曲线离散对数问题算法的进展。

第 5 章将讲述椭圆曲线密码的软件和硬件实现，还会介绍“旁门攻击”。所谓“旁门攻击”，是指敌手检测密码设备的信息泄露来获得信息的攻击方法，检测的对象包括电磁泄露、设备功耗和错误数据。

附录将给出实际工作者有用的其他信息。其中附录 A 将给出适合椭圆曲线密码使用的工作参数的具体例子。附录 B 将给出椭圆曲线密码体制的标准。附录 C 将列出相关数论计算的软件工具。

1.5 注释和进一步参考

1.1 节

关于近代密码学的流行书籍包括 Schneier 的[409]，Menezes, van Oorschot 和 Vanstone 的[319]，Stinson 的[454]，Ferguson 和 Schneier 的[136]。这些书介绍了我们在 1.1 节中讲到的对称密钥密码体制和公钥密码体制，包括对称密钥加密方案、MAC 算法、公钥加密方案、数字签名方案。用公钥密码体制开发大规模应用的实际问题在 Ford 和 Baum 的[145]，Adams 和 Lloyd 的[2]以及 Housley 和 Polk 的[200]中做了介绍。

1.2 节

公钥密码的概念是由 Diffie 和 Hellman[121]与 Merkle[321]分别独立提出的。Diffie 在[120]中详细介绍了公钥密码的历史和发展。作为一本通俗读物，请看 Levy 的书[290]。Diffie 和 Hellman 给出了一种密钥协商算法，这种算法基于素数模的整数乘法群的指数运算。他们还描述了利用一般单向陷门函数构造公钥加密和数字签名方案的理论。第一个实际的公钥密码方案是由 Merkle 和 Hellman 提出的背包方案[322]。这种背包方案及其一些变体已被证明是不安全的。

RSA 公钥加密和数字签名方案是由 Rivest, Shamir 和 Adleman 提出的[391]。

ElGamal 公钥密码是第一个基于离散对数问题困难性的公钥加密和数字签名方案[131]。在 FIPS 186 中颁布的数字签名算法[139]是由 Kravitz 设计的[268]。Smith 和 Skinner[443]，Gong 和 Harn[176]，Lenstra 和 Verheul[283]都研究了如何简洁地表示子群的元素问题，他们分别给出了 $\mathbb{F}_{p^2}^*$ 的 $p+1$ 阶子群，$\mathbb{F}_{p^3}^*$ 的 p^2+p+1 阶子群，$\mathbb{F}_{p^6}^*$ 的 p^2-p+1 阶子群的元

素的简洁表示。在他们提出的著名的 LUC, GH 和 XTR 密码体制中，分别使用了子群元素的表示法，这种表示法比有限域元素表示法分别小 2, 1.5 和 3 倍。

Koblitz[250]和 Miller[325]在 1985 年分别独立地提出了利用定义在有限域上的椭圆曲线的点群来构造离散对数密码体制的方法。有两本专门研究椭圆曲线密码的书，它们分别是 Menezes 在 1993 年出版的[313]与 Blake, Seroussi 和 Smart 在 1999 年出版的[49]。由 Enge 写的书[132]和由 Washington 写的书[474]集中研究了椭圆曲线密码的相关数学。

关于椭圆曲线的其他应用包括由Lenstra 提出的整数因子分解算法[285]（这一算法的突出优点是能快速找到整数的小素数因子），Goldwasser 和 Kilian 的算法的最初证明[173]和由 Kaliski 提出的伪随机数生成器[233]。Koyama, Maurer, Okamoto 和 Vanstone[267]证明了定义在以复合整数 n 为模的整数之上的椭圆曲线能够设计构造 RSA 类公钥密码方案，其中椭圆曲线群的阶是陷门[267]。这种 RSA 密码的安全性由分解 n 的困难性确保，因此 n 的长度应当与普通 RSA 密码的模的长度相同。Kurosawa, Okada 和 Tsujii[273], Pinch[374], Kaliski[236] 以及 Bleichenbacher[52]研究表明，这种椭圆曲线型的 RSA 密码不具有普通 RSA 密码的许多优点。

还有许多其他的基于有限群的离散对数密码方案。其中包括由 McCurley 提出的以复合整数为模的整数单位元素群[310]，由 Koblitz 提出的有限域上的超椭圆曲线雅可比行列式[251]，由 Galbraith, Paulus 和 Smart 提出的有限域上的超椭圆曲线雅可比行列式[157]，以及由 Buchmann 和 Williams 提出的虚二次数域上的类群[80]。Buchmann 和 Williams 又在[81]中证明了一个实二次数域如何产生一种能够构造离散对数方案的“类似”群的结构（可以参考 Scheidler, Buchman 和 Williams 的[405]）。Scheidler, Stein 和 Williams[406], Müller, Vanstone 和 Zuccherato[336]还研究了实同余函数域的类似结构。

1.3 节

Pollard 首先提出用于整数因子分解的数域筛法[380]。A. Lenstra 和 H. Lenstra 编辑出版了一本关于数域筛法的书[280]。Cavallar 等人[87]报告了他们利用数域筛法分解 512 位 RSA 模的工作。

Pollard 在[379]中提出了著名的 Pollard's rho 算法。利用数域筛法求解素域上的离散对数问题首先由 Gordon 提出[178]，然后 Schirokauer 又对其进行了改进[408]。Joux 和 Lercier 又做了进一步的改进，并于 2001 年用于他进行的 397 位（120 位十进制位）的离散对数问题的计算中[228]。求解二进制域离散对数问题的最快算法是由 Coppersmith 提出的[102]。而算法的实现则由 Thomé 首先进行，他于 2001 年成功地计算了 607 位二进制域 $\mathbb{F}_{2^{607}}$ 上的

对数[460]。

2002 年由 Chris Monico 领导的小组成功地解决了著名的 Certicom ECCp-109 挑战问题[88]。所用的方法是由 Van Oorschot 和 Wiener 提出的并行化 Pollard's rho 算法[463]。ECCp-109 挑战问题是要求解定义在 109 位素域上的椭圆曲线离散对数问题(ECDLP)的实例。这一工作共用了 549 天，有 10 000 多台工作站通过互联网工作。

表 1.1 中关于 ECC 的密钥长度和 DSA 的参数长度取自文件 FIPS 186-2[140]和 NIST 的专业出版物 800-56[342]。这些比较结果与 Lenstra 和 Verheul[284]以及 Lenstra[279]的建议是一致的。他们研究了成本等效密钥规模的问题。Bernstein[41], Wiener[481], Lenstra, Shamir, Tomlinson 和 Tromer[282]都提出并分析了设计降低矩阵态全成本的专用硬件装置的技术。设计降低筛态全成本的专用硬件装置的技术由 Shamir[421](也可参见 Lenstra 和 Shamir 的[281]), Geiselmann 和 Steinwandt[169], Shamir 和 Tromer[423]分别提出并进行了研究。Shamir 和 Tromer[423]对利用筛法分解 1024 位的 RSA 模的成本进行了评估，评估认为用 1000 万美元制造的专用硬件设备用不到 1 年的时间可以完成，并且对于矩阵态将更容易。

1.4 节

读者通过研究密码学年会的会议录可 以及时了解椭圆曲线密码及相关领域的前沿。主要的密码学年会有 ASIACRYPT, CRYPTO, EUROCRYPT, INDOCRYPT 等。此外还有密码学硬件与嵌入式系统(CHES)专题讨论会、公钥密码理论(PKC)与实践国际专题讨论会、两年一次的算法数论讨论会(ANTS)等。所有这些会议的会议录都由 Springer-Verlag 出版社出版，并纳入到了它的 *Lecture Notes in Computer Science* 系列。通过网络很容易查到，网址为 http://link.springer.de/link/service/series/0558/。另外，关于密码学最新研究论文的重要知识库是密码学电子论文档案网站，网址为 http://eprint.iacr.org/。

第 2 章　有限域算术

有限域算术运算的有效实现是有效实现椭圆曲线密码的重要的先决条件，因为椭圆曲线的运算是基于有限域的算术运算的。2.1 节将首先简单介绍有限域的理论。椭圆曲线密码体制的有效实现主要用到三种类型的有限域，它们是素域、二进制域和最佳扩域。这几种域的加、减、乘、求逆运算的软件实现有效算法，将分别在 2.2 节、2.3 节和 2.4 节中讲述。其硬件实现将在 5.2 节和本章的 2.5 节中讲述。

2.1　有限域介绍

域是对常见的数系（如有理数$\mathbb{Q}$、实数$\mathbb{R}$和复数$\mathbb{C}$）及其基本特性的抽象。域由一个集合$\mathbb{F}$和两种运算共同组成，这两种运算分别为加法（用$+$表示）和乘法（用$\cdot$表示），并且满足下列算术特性：

(i) $(\mathbb{F},+)$对于加法运算构成加法交换群，单位元用 0 表示。

(ii) $(\mathbb{F}\setminus\{0\},\cdot)$对于乘法运算构成乘法交换群，单位元用 1 表示。

(iii) 分配律成立：对于所有的$a,b,c\in\mathbb{F}$，都有$(a+b)\cdot c=a\cdot c+b\cdot c$。

若集合$\mathbb{F}$是有限集合，则称域为有限域。

本节介绍有限域的基本概念，其他性质将在书中需要时再做介绍。

2.1.1　域运算

一个域$\mathbb{F}$具有两种运算，即加法和乘法。域元素的减法用加法来定义：对于$a,b\in\mathbb{F}$，$a-b=a+(-b)$，其中$-b$是b的负元素，它是使得$b+(-b)=0$的惟一的一个域元素。类似地，域元素的除法用乘法来定义：对于$a,b\in\mathbb{F}$，$b\neq 0$，$a/b=a\cdot b^{-1}$，其中b^{-1}是b的逆元素，它是使得$b\cdot b^{-1}=1$的惟一的一个域元素。

2.1.2　存在性和惟一性

有限域的元素的个数称为有限域的阶。存在一个 q 阶有限域 $\mathbb{F}$，当且仅当 q 是一个素数的幂，即 $q = p^m$，其中 p 是一个素数并称为域 $\mathbb{F}$ 的特征，m 是一个正整数。若 $m = 1$，则域 $\mathbb{F}$ 就称为素域。若 $m \geqslant 2$，则域 $\mathbb{F}$ 就称为扩域。对于任意一个素数幂 q，本质上只存在一个 q 阶有限域。这意味着，任意两个 q 阶有限域除了域元素的表示符号不同外，在结构上它们是相同的（参考例 2.3）。我们称任意两个 q 阶有限域都同构，并用符号 $\mathbb{F}_q$ 表示它们。

2.1.3　素域

设 p 是一个素数，以 p 为模，则模 p 的全体余数的集合 $\{0, 1, 2, \ldots, p-1\}$ 关于模 p 的加法和乘法构成一个 p 阶有限域，并用符号 $\mathbb{F}_p$ 表示。我们称 p 为 $\mathbb{F}_p$ 的模。对于任意的整数 a，$a \bmod p$ 表示用 p 除 a 所得到的余数 r，$0 \leqslant r \leqslant p-1$，并称这种运算为求模 p 的剩余。

例 2.1　（素域 $\mathbb{F}_{29}$）$\mathbb{F}_{29}$ 的元素是 $\{0, 1, 2, \ldots, 28\}$。下面是域 $\mathbb{F}_{29}$ 的一些算术运算的例子。

(i) 加法：$17 + 20 = 8$，因为 $37 \bmod 29 = 8$。

(ii) 减法：$17 - 20 = 26$，因为 $-3 \bmod 29 = 26$。

(iii) 乘法：$17 \cdot 20 = 21$，因为 $340 \bmod 29 = 21$。

(iv) 求逆：$17^{-1} = 12$，因为 $17 \cdot 12 \bmod 29 = 1$。

2.1.4　二进制域

阶为 2^m 的域称为二进制域或特征为 2 的有限域。构成 $\mathbb{F}_{2^m}$ 的一种方法是采用多项式基表示法。在这种表示法中，域 $\mathbb{F}_{2^m}$ 的元素是次数最多为 $m-1$ 次的二进制多项式（多项式的系数取自 $\mathbb{F}_2 = \{0, 1\}$）：

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 : a_i \in \{0, 1\}\}。$$

选择一个二进制域上的 m 次既约多项式 $f(z)$（对于任意的 m，这样的多项式一定存在，并能有效产生，参见附录 A.1）。$f(z)$ 的既约性意味着 $f(z)$ 不能被分解成次数低于 m 的二进制域上的多项式的乘积。域元素的加法是两个多项式系数的模 2 相加。域元素的乘法是两个多项式相乘后取模 $f(z)$。对于任意一个二进制域上的多项式 $a(z)$，$a(z) \bmod f(z)$ 表示一个次数低于 m 的余式多项式 $r(z)$。余式多项式 $r(z)$ 可用 $f(z)$ 辗转相除 $a(z)$ 得到，这一运算称为求模 $f(z)$ 的余式。

例 2.2 （二进制域$\mathbb{F}_{2^4}$）域$\mathbb{F}_{2^4}$由下面 16 个次数小于 4 的多项式组成：

0	z^2	z^3	z^3+z^2
1	z^2+1	z^3+1	z^3+z^2+1
z	z^2+z	z^3+z	z^3+z^2+z
$z+1$	z^2+z+1	z^3+z+1	z^3+z^2+z+1

下面列出约减多项式$f(z)=z^4+z+1$的域$\mathbb{F}_{2^4}$上的一些算术运算：

(i) 加法：$(z^3+z^2+1)+(z^2+z+1)=z^3+z$。

(ii) 减法：$(z^3+z^2+1)-(z^2+z+1)=z^3+z$。（注意，因为在$\mathbb{F}_2$上$-1=1$，所以对于所有$a\in\mathbb{F}_{2^m}$都有$-a=a$。）

(iii) 乘法：$(z^3+z^2+1)\cdot(z^2+z+1)=z^2+1$。因为

$$(z^3+z^2+1)\cdot(z^2+z+1)=z^5+z+1$$

和

$$(z^5+z+1) \bmod (z^4+z+1)=z^2+1。$$

(iv) 求逆：$(z^3+z^2+1)^{-1}=z^2$，因为$(z^3+z^2+1)\cdot z^2 \bmod (z^4+z+1)=1$。

例 2.3 （域的同构）设有三个二进制域上次数为 4 的既约多项式$f_1(z)=z^4+z+1$，$f_2(z)=z^4+z^3+1$和$f_3(z)=z^4+z^3+z^2+z+1$。这三个约减多项式都可用于构造域$\mathbb{F}_{2^4}$，分别记为K_1，K_2，K_3。域K_1，K_2，K_3都由相同的十六个次数小于 4 的多项式构成。特别地，这三个域的表现形式不同。如在K_1中$z^3\cdot z=z+1$，在K_2中$z^3\cdot z=z^3+1$，而在K_3中$z^3\cdot z=z^3+z^2+z+1$。然而，所有阶数相等的域都是同构的，即它们的惟一不同是域元素的表示形式不同。可以如下构造K_1和K_2之间的一个同构映射，寻找一个元素$c\in K_2$，使得$f_1(c)\equiv 0 \pmod{f_2}$，并将映射$z\mapsto c$引申成为同构映射$\varphi$：$K_1\to K_2$；这样选择的$c$就是$z^2+z, z^2+z+1, z^3+z^2$和$z^3+z^2+1$。

2.1.5 扩域

二进制域的多项式基表示法可以按如下方法推广到所有的扩域。设p是一个素数，$m\geqslant 2$。令$\mathbb{F}_p[z]$表示所有系数取自$\mathbb{F}_p$上的z的多项式的集合。令$f(z)$是$\mathbb{F}_p[z]$上的一个m次既约多项式，以$f(z)$作为约减多项式。对于任意的p和m，这样的既约多项式一定存在，而且能够有效地产生（参见附录 A.1）。$f(z)$的既约性意味着它不能分解为$\mathbb{F}_p[z]$上次数小于m的多项式的乘积。域$\mathbb{F}_{p^m}$上的元素都是域$\mathbb{F}_p[z]$上次数小于m的多项式：

$$\mathbb{F}_{p^m}=\{a_{m-1}z^{m-1}+a_{m-2}z^{m-2}+\cdots+a_2z^2+a_1z+a_0 : a_i\in\mathbb{F}_p\}。$$

域的加法就是多项式的系数按 $\mathbb{F}_p$ 的运算规则相加。域的乘法就是模 $f(z)$ 的多项式乘。

例 2.4（扩域）设 $p=251$，$m=5$。多项式 $f(z)=z^5+z^4+12z^3+9z^2+7$ 是 $\mathbb{F}_{251}[z]$ 上的既约多项式，以 $f(z)$ 作为约减多项式构造域 $\mathbb{F}_{251^5}$。$\mathbb{F}_{251^5}$ 的阶为 251^5，其元素都是 $\mathbb{F}_{251^5}[z]$ 上的次数小于等于 4 的多项式。

下面给出域 $\mathbb{F}_{251^5}$ 上的一些算术运算。令 $a=123z^4+76z^2+7z+4$，$b=196z^4+12z^3+225z^2+76$。

(i) 加法：$a+b=68z^4+12z^3+50z^2+7z+80$。

(ii) 减法：$a-b=178z^4+239z^3+102z^2+7z+179$。

(iii) 乘法：$a\cdot b=117z^4+151z^3+117z^2+182z+217$。

(iv) 求逆：$a^{-1}=109z^4+111z^3+250z^2+98z+85$。

2.1.6　有限域的子域

设 k 是域 K 的子集合，且 k 对于 K 中的运算构成域，则称 k 为 K 的子域，称 K 是 k 的扩域。有限域的子域容易表征。对于 m 的每一个正因子 l，有限域 $\mathbb{F}_{p^m}$ 恰好有阶为 p^l 的子域，且子域的元素都是域 $\mathbb{F}_{p^m}$ 上满足 $a^{p^l}=a$ 的元素。反过来说，域 $\mathbb{F}_{p^m}$ 的每一个子域的阶都是 p^l，l 是 m 的正因子。

2.1.7　有限域的基

有限域 $\mathbb{F}_{q^n}$ 可以看做是域 $\mathbb{F}_q$ 上的一个向量空间。其向量是域 $\mathbb{F}_{q^n}$ 的元素，标量是域 $\mathbb{F}_q$ 的元素，向量的加法运算是域 $\mathbb{F}_{q^n}$ 上的加法运算，标量与向量的乘法是域 $\mathbb{F}_q$ 的元素与域 $\mathbb{F}_{q^n}$ 的元素在域 $\mathbb{F}_{q^n}$ 上的乘法运算。这个向量空间的维数是 n，并有许多基底（简称为基）。

若 $B=\{b_1,b_2,\ldots,b_n\}$ 是一组基，则域 $\mathbb{F}_{q^n}$ 上的元素 a 可惟一地表示为域 $\mathbb{F}_q$ 上的一个 n 维向量 $(a_1,a_2,\ldots,a_n)$，其中 $a=a_1b_1+a_2b_2+\cdots+a_nb_n$。例如，在域 $\mathbb{F}_{p^m}$ 的多项式基表示中，域 $\mathbb{F}_{p^m}$ 是域 $\mathbb{F}_p$ 上的一个 m 维向量空间，并且 $\{z^{m-1},z^{m-2},\ldots,z^2,z,1\}$ 是一组基。

2.1.8　有限域的乘法群

有限域 $\mathbb{F}_q$ 的全体非零元素构成一个乘法循环群，记为 $\mathbb{F}_q^*$。$\mathbb{F}_q^*$ 中存在一个元素 b，b 称为生成元，使得

$$\mathbb{F}_q^*=\{b^i:0\leqslant i\leqslant q-2\}。$$

$a\in\mathbb{F}_q^*$ 的阶是满足 $a^t=1$ 的正整数 t。因为 $\mathbb{F}_q^*$ 是循环群，所有 t 是 $q-1$ 的因子。

2.2 素域算术

本节给出素域$\mathbb{F}_p$的算术运算算法。任意素数p的运算算法在2.2.1节至2.2.5节中给出。当模数p具有特殊形式时，其约减运算的速度大大提高。NIST素数（如$p=2^{192}-2^{64}-1$）的有效约减运算算法在2.2.6节中给出。

下面我们将给出的一种算法特别适合软件实现。我们假设实现平台的字长为W位，W是8的整数倍。通常工作站的字长是64位或32位。而低档或廉价的设备的字长W较小，如某些嵌入式系统的字长是16位，而智能卡的字长可能只有8位。W位的字U的各位依次编号为0到$W-1$，最右边的位为第0位。

域$\mathbb{F}_p$的元素是从0到$p-1$的整数。令$m=\lceil \mathrm{lb}\, p\rceil$是$p$的二进制长度，$t=\lceil m/W\rceil$是表示该域元素所用的$W$位字的个数。图2.1显示域元素$a$以二进制形式存储在一个$t$个$W$位字的数组$A=(A[t-1],\ldots,A[2],A[1],A[0])$中，其中最右边的$A[0]$是最低有效位。

$A[t-1]$	$\cdots$	$A[2]$	$A[1]$	$A[0]$

图2.1 将域$\mathbb{F}_p$的元素a表示为W位字的数组A，

$$a=2^{(t-1)W}A[t-1]+\cdots+2^{2W}A[2]+2^{W}A[1]+A[0]$$

适合硬件实现的方法与这里给出的算法和域元素表示法不同。5.1.1节分析了乘法运算中可能的瓶颈是硬件的整数乘和进位的传播。5.1.2节简单讨论了浮点运算硬件的应用，在工作站中通常都有浮点运算硬件。这种浮点运算硬件可以大大提高乘法的速度（采用不同的域元素表示方法）。类似地，5.1.3节介绍了应用某些处理器中的单指令多数据(SIMD)寄存器。5.1.5节介绍了域运算的时序选择。

2.2.1 加法和减法

域的加法和减法的运算算法以多字整数算法的形式给出。算法中使用了如下的符号和术语：整数w的赋值操作“$(\varepsilon,z)\leftarrow w$”表示

$$z\leftarrow w \bmod 2^W，\text{和}$$
$$\varepsilon\leftarrow 0，\text{若}\ w\in[0,2^W)，\text{否则，}\ \varepsilon\leftarrow 1。$$

若对于$x,y\in[0,2^W)$和$\varepsilon'\in\{0,1\}$，有$w=x+y+\varepsilon'$，则$w=\varepsilon 2^W+z$，并称ε为单字加的进位（当且仅当$z<x+\varepsilon'$时有$\varepsilon=1$）。算法2.5用于实现多字整数加法。

算法 2.5　多精度加法

输入：整数 a，$b \in [0, 2^{Wt})$。

输出：(ε, c)，其中 $c = a + b \mod 2^{Wt}$，ε是进位。

1. $(\varepsilon, C[0]) \leftarrow A[0] + B[0]$。
2. 对于 i 从 1 到 $t-1$，重复执行
 2.1　$(\varepsilon, C[i]) \leftarrow A[i] + B[i] + \varepsilon$。
3. 返回(ε, c)。

对于指令能自动处理进位的处理器，就不必要另外处理进位。与加法类似，算法2.6 给出了多字减法算法，对于减法通常不称进位而称为借位。

算法 2.6　多精度减法

输入：整数 a，$b \in [0, 2^{Wt})$。

输出：(ε, c)，其中 $c = a - b \mod 2^{Wt}$，ε是借位。

1. $(\varepsilon, C[0]) \leftarrow A[0] - B[0]$。
2. 对于 i 从 1 到 $t-1$，重复执行
 2.1　$(\varepsilon, C[i]) \leftarrow A[i] - B[i] - \varepsilon$。
3. 返回(ε, c)。

对算法 2.5 和算法 2.6 进行修改便可得到模加$((x + y) \mod p)$和模减$((x - y) \mod p)$的算法。

算法 2.7　域 $\mathbb{F}_p$上的加法

输入：模数 p 和整数 a，$b \in [0, p-1]$。

输出：$c = (a + b) \mod p$。

1. 用算法 2.5 得到 (ε, c)，其中 $c = a + b \mod 2^{Wt}$，ε 是进位。
2. 若$\varepsilon = 1$，则把$c = (C[t-1], \ldots, C[2], C[1], C[0])$减 p；
 否则，若$c \geqslant p$，则$c \leftarrow c - p$。
3. 返回(c)。

算法 2.8　域 $\mathbb{F}_p$上的减法

输入：模数 p 和整数 a，$b \in [0, p-1]$。

输出：$c = (a - b) \mod p$。

1. 用算法 2.6 得到 (ε, c)，其中 $c = a - b \bmod 2^{Wt}$，ε 是借位。
2. 若 $\varepsilon = 1$，则把 $c = (C[t-1], \ldots, C[2], C[1], C[0])$ 加 p。
3. 返回(c)。

2.2.2 整数乘

域 $\mathbb{F}_p$ 上的乘法 $a \cdot b$ 可以通过先将 a 和 b 作为整数乘，然后取模 p 来完成。算法 2.9 和算法 2.10 分别给出了基本的整数乘法算法，它们分别阐明了基本运算数扫描和乘积扫描的方法。在两个算法中，(UV)表示一个$(2W)$位的量，这个量是 W 位的字 U 和 V 的级联。

算法 2.9 整数乘（运算数扫描方式）

输入：整数 $a, b \in [0, p-1]$。

输出：$c = a \cdot b$。

1. 对于 $0 \leqslant i \leqslant t-1$，令 $C[i] \leftarrow 0$。
2. 对于 i 从 0 到 $t-1$，重复执行
 2.1 $U \leftarrow 0$。
 2.2 对于 j 从 0 到 $t-1$，重复执行
 $(UV) \leftarrow C[i+j] + A[i] \cdot B[j] + U$。
 $C[i+j] \leftarrow V$。
 2.3 $C[i+t] \leftarrow U$。
3. 返回(c)。

在步骤 2.2 计算 $C[i+j] + A[i] \cdot B[j] + U$ 的运算称为求中间积。因为运算数是 W 位的数，所以乘积的上界为 $2(2^W - 1) + (2^W - 1)^2 = 2^{2W} - 1$，并能用符号 (UV) 表示。

算法 2.10 的编排使 $c = ab$ 的计算从右向左进行。与前一个算法一样，两个 W 位的数相乘，要产生一个$(2W)$位的乘积。下面的算法中，R_0, R_1, R_2, U 和 V 都是 W 位的字。

算法 2.10 整数乘（积扫描方式）

输入：整数 a, $b \in [0, p-1]$。

输出：$c = a \cdot b$。

1. $R_0 \leftarrow 0$, $R_1 \leftarrow 0$, $R_2 \leftarrow 0$。
2. 对于 k 从 0 到 $2t-2$，重复执行
 2.1 对于集合 $\{(i, j) \mid i + j = k, 0 \leqslant i, j \leqslant t-1\}$ 中的每一个元素，重复执行

$(UV) \leftarrow A[i] \cdot B[j]$。

$(\varepsilon, R_0) \leftarrow R_0 + V$。

$(\varepsilon, R_1) \leftarrow R_1 + U + \varepsilon$。

$R_2 \leftarrow R_2 + \varepsilon$。

2.2　$C[k] \leftarrow R_0$, $R_0 \leftarrow R_1$, $R_1 \leftarrow R_2$, $R_2 \leftarrow 0$。

3. $C[2t-1] \leftarrow R_0$。

4. 返回(c)。

注 2.11（*算法 2.9 和算法 2.10 的实现*）算法 2.9 和算法 2.10 被描述成适合这样的环境，即 W 位的乘法运算产生 $2W$ 位的乘积结果（如 Intel 的 Pentium 或 Sun SPARC）。一个例外是 64 位的 Sun UltraSPARC，它的乘法器对于 64 位的输入产生低 64 位乘积。一种变形算法将 a 和 b 分成$(W/2)$位的半字，但却用 W 位的寄存器进行累加计算。5.1.3 节还将介绍一个例子，它是 32 位的处理器，却有某些 64 位的运算。

Karatsuba-Ofman 乘

对于两个 n 位的整数相乘，算法 2.9 和算法 2.10 需要$O(n^2)$次位运算。一种由 Karatsuba 和 Ofman 提出的“分而治之”算法将计算复杂性减少到了 $O(n^{\lg 3})$。设 $n = 2l$，$x = x_1 2^l + x_0$，$y = y_1 2^l + y_0$，它们都是 $2l$ 位的整数。则

$$\begin{aligned} xy &= (x_1 2^l + x_0)(y_1 2^l + y_0) \\ &= x_1 \cdot y_1 2^{2l} + [(x_0 + x_1) \cdot (y_0 + y_1) - x_1 y_1 - x_0 \cdot y_0] 2^l + x_0 y_0, \end{aligned}$$

xy 通过 3 次 l 位整数乘法（与普通 $2l$ 位整数相乘不同）、两次加法和两次减法完成[①]。对于大的 l，加法和减法的时间消耗远小于乘法。这一过程可以对中间值递归调用，在某一极限条件（比如到达机器所能表示的最大值）时结束，再采用其他传统方法。

对于适当大小的整数，Karatsuba-Ofman 算法是很重要的。为了减少移位次数（用 2^l 和 2^{2l} 乘）和使用更高效的面向字的操作，算法的实际实现可能与传统描述不完全一样。例如，打破字的界限可能会更有效，并且可以把给定的数位分裂成两个以上的更小的数位。

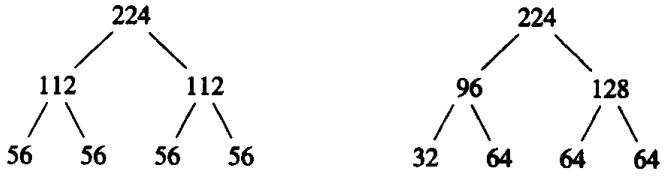
例 2.12（Karatsuba-Ofman *方法*）考虑两个 224 位的数 x 和 y 相乘，机器的字长 $W = 32$。两种深度为 2 的分裂方法示于图 2.2 中。图 2.2(a)中的分裂方法在数学上是很优雅的，而且与图 2.2(b)相比有更多的可重用代码。然而却需要更多的移位（因为这种分裂不是按字的界

① 截项可以写成$(x_0 - x_1)(y_1 - y_0) + x_0 y_0 + x_1 y_1$，若事先知道 $x_0 \geqslant x_1$ 且 $y_1 \geqslant y_0$，则对某些平台可能是有用的。

进行的)。如果56位的量相乘(可能是Karatsuba-Ofman方法的另一种应用)与64位的量相乘有近似相同的资源消耗,则该分裂法较为节省硬件的计算能力,因为按图2.2(a)的方法需要9次64位乘,而按图2.2(b)中的方法需要1次32位乘和8次64位乘。另一方面,图2.2(b)中的按字的界进行分裂要计算更复杂的截项,因为在字的加法中可能有进位。例如,深度为2的截项具有以下形式:

$$(x_0+x_1)(y_0+y_1)-x_1y_1-x_0y_0,$$

其中x_0+x_1和y_0+y_1在图2.2(a)中是57位,而在图2.2(b)中是65位。图2.2(b)所示的分裂方法的成本稍微高一些,虽然$(x_0+x_1)(y_0+y_1)$可以当做64×64加上两个与高位对应的加法。



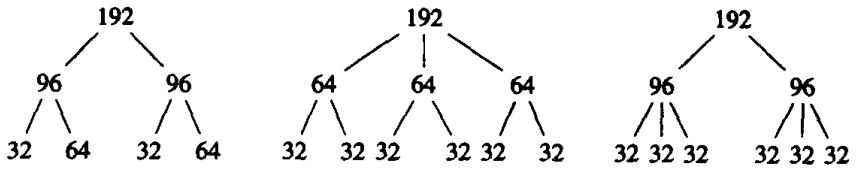
(a) $n/2$ 分裂　　(b) 按字界分裂

图 2.2　224位整数的深度为2的分裂法。计算乘积xy,采用图2.2(a)的方法需要3次112×112,每一次都用3个56×56完成。而采用图2.2(b)的方法需要1次96×96(分裂为1次32×32和2次64×64)和2次128×128(每次都导致3次64×64)

现在用另外一种方法描述将Karatsuba-Ofman方法用于两个192位数的乘法,机器字W长为32位。在图2.3中给出了三种可能的深度为2的分裂方法。图2.3(a)的分裂方法需要计算21次32×32,而图2.3(b)和图2.3(c)的分裂方法都只需要计算18次32×32。这种乘法的基本思想如下。设x和y是两个$3l$位的整数,$x=x_2 2^{2l}+x_1 2^l+x_0$,$y=y_2 2^{2l}+y_1 2^l+y_0$,则x乘y可表示为

$$\begin{aligned}xy&=(x_2 2^{2l}+x_1 2^l+x_0)\cdot(y_2 2^{2l}+y_1 2^l+y_0)\\&=x_2y_2 2^{4l}+(x_2y_1+x_1y_2)2^{3l}+(x_2y_0+x_0y_2+x_1y_1)2^{2l}\\&\quad+(x_1y_0+x_0y_1)2^l+x_0y_0\\&=x_2\cdot y_2 2^{4l}+[(x_2+x_1)\cdot(y_2+y_1)-x_2y_2-x_1\cdot y_1]2^{3l}\\&\quad+[(x_2+x_0)\cdot(y_2+y_0)-x_2y_2-x_0\cdot y_0+x_1y_1]2^{2l}\\&\quad+[(x_1+x_0)\cdot(y_1+y_0)-x_1y_1-x_0y_0]2^l+x_0y_0,\end{aligned}$$

总共需要6次l位整数乘法。



(a)二叉分裂　　(b)深度为 1 的三叉分裂　　(c)深度为 2 的三叉分裂

图 2.3　192 位数的深度为 2 的分裂。计算乘积 xy，采用图(a)的方法需要 3 次 96×96，每一次都用一个 32 × 32 和两个 64 × 64（每一次需要 3 个 32 × 32），总共需要 21 次 32 × 32。采用图(b)和图(c)的方法都只需要 18 次 32 × 32

有限域的乘法是椭圆曲线密码体制的基本运算。硬件整数乘法器和进位传播可能成为算法 2.9 和算法 2.10 直接实现的瓶颈。2.2 节开始时我们就指出，第 5 章将讨论适用于某些环境的可选方案。

2.2.3　整数平方

域 $\mathbb{F}_p$ 上的元素 a 的平方可以这样计算，首先把 a 作为整数计算出平方，然后求模 p 的余数。对算法 2.10 进行直接修改便得到如下的求整数平方的算法，减少了约一半的单精度乘法。在步骤 2.1 中，2 与 $(2W)$ 位的量 (UV) 相乘，得到 $(2W+1)$ 位的结果 (ε, UV)。

算法 2.13　整数平方

输入：整数 $a \in [0, p-1]$。

输出：$c = a^2$。

1. $R_0 \leftarrow 0, R_1 \leftarrow 0, R_2 \leftarrow 0$。
2. 对于 k 从 0 到 $2t-2$，重复执行
 2.1　对于集合 $\{(i,j) \mid i+j=k, 0 \leqslant i \leqslant j \leqslant t-1\}$ 中的每一个元素，重复执行
 $(UV) \leftarrow A[i] \cdot A[j]$。
 若（$i<j$），则 $(\varepsilon, UV) \leftarrow (UV) \cdot 2, R_2 \leftarrow R_2 + \varepsilon$。
 $(\varepsilon, R_0) \leftarrow R_0 + V$。
 $(\varepsilon, R_1) \leftarrow R_1 + U + \varepsilon$。
 $R_2 \leftarrow R_2 + \varepsilon$。
 2.2　$C[k] \leftarrow R_0, R_0 \leftarrow R_1, R_1 \leftarrow R_2, R_2 \leftarrow 0$。
3. $C[2t-1] \leftarrow R_0$。
4. 返回(c)。

算法中步骤 2.1 的用 2 乘可以通过两个单精度带进位移位（如果可能的话）或两个单精度带进位加法来实现。该步可以重新改写，以使每个输出字$C[k]$最多需要一次用 2 乘，但这样需要两个累加器和相应的累加步骤。

2.2.4 约减

对于非特殊形式的模 p，求 $z \bmod p$ 的计算是模乘计算中费时的部分。因为椭圆曲线密码体制的性能取决于域乘法的速度，因此选择模就非常重要。例如可选择 2.2.6 节中 NIST 推荐的素数，这些素数的模运算非常快。本节仅给出 Barrett 约减的方法和 Montgomery 乘法的概念。

Barrett 和 Montgomery 方法都用低成本的运算代替了高成本的除法。可以用 Barrett 约减方法直接替换传统的约减方法，然而需要一种高成本的与模相关的计算；因此这种方法适合于对于一个模的多次约减运算。另一方面，Montgomery 方法需要对数据进行变换。当多次中间乘法所节省的的收益大于输入输出数据变换的消耗时，这种方法是有效的，例如求模幂的运算。

注意，如 4.4 节介绍的数字签名那样的大系统中需要某些模运算，并且不采用特殊形式的模。在这些实例中采用 Barrett 约减方法可能是合适的。

Barrett 约减

Barrett 约减方法（算法 2.14）对于给定的正整数 z 和 p 求 $z \bmod p$。与 2.2.6 节给出的算法不同，Barrett 约减方法不使用任何特殊的模 p。用一种低成本的求商 $\lfloor z/p \rfloor$ 运算，这种运算采用适当选择的基 b（如 $b=2^L$，L 根据模来选择，而不是根据 z 来选择）的幂。要计算一个与模相关的量 $\lfloor b^{2k}/p \rfloor$，以使算法适合对同一个模进行多次约减的运算。

算法 2.14 Barrett 约减

输入：整数 $p, b \geqslant 3$，$k=\lfloor \log_b p \rfloor + 1$，$0 \leqslant z < b^{2k}$，$\mu = \lfloor b^{2k}/p \rfloor$。

输出：$z \bmod p$。

1. $\hat{q} \leftarrow \left\lfloor \lfloor z/b^{k-1} \rfloor \cdot \mu / b^{k+1} \right\rfloor$。
2. $r \leftarrow (z \bmod b^{k+1}) - (\hat{q} \cdot p \bmod b^{k+1})$。
3. 若 $r<0$，则 $r \leftarrow r + b^{k+1}$。
4. 当 $r \geqslant p$ 时，重复执行 $r \leftarrow r - p$。
5. 返回(r)。

注 2.15（算法 2.14 的正确性）令 $q=\lfloor z/p\rfloor$，则 $r=z \bmod p=z-qp$。算法 2.14 的步骤 1 计算 q 的一个估值 $\hat{q}$，因为

$$\frac{z}{p}=\frac{z}{b^{k-1}}\cdot\frac{b^{2k}}{p}\cdot\frac{1}{b^{k+1}}。$$

注意，

$$0\leqslant \hat{q}=\left\lfloor\frac{\left\lfloor\frac{z}{b^{k-1}}\right\rfloor\cdot\mu}{b^{k+1}}\right\rfloor\leqslant\left\lfloor\frac{z}{p}\right\rfloor=q。$$

下列的推证表明 $q-2\leqslant \hat{q} \leqslant q$，即 $\hat{q}$ 是 q 的一个很好的估值。定义

$$\alpha=\frac{z}{b^{k-1}}-\left\lfloor\frac{z}{b^{k-1}}\right\rfloor,\quad \beta=\frac{b^{2k}}{p}-\left\lfloor\frac{b^{2k}}{p}\right\rfloor。$$

则 $0\leqslant \alpha, \beta<1$ 且

$$q=\left\lfloor\frac{\left(\left\lfloor\frac{z}{b^{k-1}}\right\rfloor+\alpha\right)\left(\left\lfloor\frac{b^{2k}}{p}\right\rfloor+\beta\right)}{b^{k+1}}\right\rfloor$$

$$\leqslant\left\lfloor\frac{\left\lfloor\frac{z}{b^{k-1}}\right\rfloor\cdot\mu}{b^{k+1}}+\frac{\left\lfloor\frac{z}{b^{k-1}}\right\rfloor+\left\lfloor\frac{b^{2k}}{p}\right\rfloor+1}{b^{k+1}}\right\rfloor。$$

因为 $z<b^{2k}$ 且 $p\geqslant b^{k-1}$，由此得出

$$\left\lfloor\frac{z}{b^{k-1}}\right\rfloor+\left\lfloor\frac{b^{2k}}{p}\right\rfloor+1\leqslant(b^{k+1}-1)+b^{k+1}+1=2b^{k+1}$$

和

$$q\leqslant\left\lfloor\frac{\left\lfloor\frac{z}{b^{k-1}}\right\rfloor\cdot\mu}{b^{k+1}}+2\right\rfloor=\hat{q}+2。$$

在步骤 2 计算的值 r 必须满足 $r\equiv z-\hat{q}p \pmod{b^{k+1}}$，而 $|r|<b^{k+1}$。因此在步骤 3 后，$0\leqslant r<b^{k+1}$ 且 $r=z-\hat{q}p \bmod b^{k+1}$。现因 $0\leqslant z-qp<p$，我们得到

$$0\leqslant z-\hat{q}p\leqslant z-(q-2)p<3p。$$

因为 $b\geqslant 3$ 且 $p<b^k$，我们有 $3p<b^{k+1}$。于是 $0\leqslant z-\hat{q}p<b^{k+1}$，并且在步骤 3 后有 $r=z-\hat{q}p$。所以，为了获得 $0\leqslant r<p$ 而有 $r=z \bmod p$，在步骤 4 中最多需要两次减法。

注 2.16 （从计算角度分析算法 2.14）

(i) 对于基的自然选择是$b=2^L$，其中L接近处理器的字长。

(ii) 与μ的计算不同（根据模一次完成），除法运算需要以b为基的简单移位。

(iii) 令$z'=\lfloor z/b^{k-1}\rfloor$。注意，$z'$和$\mu$最多有$k+1$位基为$b$的数字。在算法的步骤 1 计算$\hat{q}$时丢掉积$z'\mu$的$k+1$个最低位。给出基为$b$的表示式$z'=\sum z'_i b^i$和$\mu=\sum \mu_j b^j$，于是

$$z'\mu=\sum_{l=0}^{2k}\underbrace{\Big(\sum_{i+j=l} z'_i\mu_j\Big)}_{w_l}b^l,$$

其中w_l可能超过$b-1$。若$b\geqslant k-1$，则$\sum_{l=0}^{k-2} w_l b^l < b^{k+1}$，因此

$$0\leqslant \frac{z'\mu}{b^{k+1}}-\sum_{l=k-1}^{2k}\frac{w_l b^l}{b^{k+1}}=\sum_{l=0}^{k-2}\frac{w_l b^l}{b^{k+1}}<1。$$

由此得出$\lfloor \sum_{l=k-1}^{2k} w_l b^l / b^{k+1}\rfloor$与$\hat{q}$最多差1，如果$b\geqslant k-1$。最多$\binom{k+2}{2}+k=(k^2+5k+2)/2$次单精度乘（值小于$b$的乘）需要找出这种对$\hat{q}$的评估。

(iv) 在步骤 2 仅需要$k+1$个$\hat{q}\cdot p$的低有效位。因为$p<b^k$，所以通过$\binom{k+1}{2}+k$次单精度乘就可获得这$k+1$个数字。

Montgomery 乘法

与 Barrett 约减类似，Montgomery 方法的思想是在古典的约减算法中用简单的运算代替除运算。这一方法对单个模乘并没有效，但是用它计算模幂却非常有效，因为计算模幂时要完成给定输入的多次模乘。本节我们仅仅给出 Montgomery 乘法的一般介绍（更详细的内容请参看 2.5 节）。

令$R>p$且$\gcd(R,p)=1$。Montgomery 约减运算是对一个输入$z<pR$，产生$zR^{-1}\bmod p$。我们考虑p是奇数的情况，以使$R=2^{Wt}$可以被选用，并且用R去除也相对容易。若$p'=-p^{-1}\bmod R$，则通过计算

$$c\leftarrow(z+(zp'\bmod R)p)/R,$$
$$若\ c\geqslant p,\ 则 c\leftarrow c-p,$$

可以得到$c=zR^{-1}\bmod p$。其中共用了$t(t+1)$次单精度乘法，并且没有用除法。

给定 $x \in [0, p)$，令 $\tilde{x} = xR \mod p$。注意 $(\tilde{x}\tilde{y})R^{-1} \bmod p = (xy)R \mod p$。即 Montgomery 约减可用于 $\tilde{x}$ 表示的乘法中。我们定义 $\tilde{x}$ 和 $\tilde{y}$ 的 Montgomery 积为

$$\mathrm{Mont}(\tilde{x}, \tilde{y}) = \tilde{x}\tilde{y}R^{-1} \bmod p = xyR \bmod p。 \tag{2.1}$$

对于单个模乘运算，采用复杂的变换 $x \mapsto \tilde{x} = xR \bmod p$ 和 $\tilde{x} \mapsto \tilde{x}R^{-1} \bmod p = x$ 是不合算的。然而，对于需要进行大量的模乘计算时只进行一次数据变换是完全可以接受的，如计算模幂。算法 2.17 给出了利用 Montgomery 方法计算模幂的算法。

算法 2.17　Montgomery 幂（基本的）

输入：奇数模 p，$R = 2^{Wt}$，$p' = -p^{-1} \bmod R$，$x \in [0, p)$，$e = (e_l, \ldots, e_0)_2$。

输出：$x^e \bmod p$。

1. $\tilde{x} \leftarrow xR \bmod p$，$A \leftarrow R \bmod p$。
2. 对于 i 从 l 到 0，重复执行
 2.1　$A \leftarrow \mathrm{Mont}(A, A)$。
 2.2　若 $e_i = 1$，则 $A \leftarrow \mathrm{Mont}(A, \tilde{x})$。
3. 返回（$\mathrm{Mont}(A, 1)$）。

进行一下粗略的比较，Montgomery 约减需要 $t(t+1)$次单精度乘法，而 Barrett 方法（对于 $b = 2^W$）却需要 $t(t+4)+1$ 次。因此，利用 Montgomery 方法计算模幂是十分有效的。这两种方法都慢于 2.2.6 节介绍的特殊模的直接约减技术。

Montgomery 算术还用来加速模逆的计算，因为在模逆计算中需要反复进行乘法。通过计算 $a^{p-2} \bmod p$ 来求出 a^{-1}，因为当 $\gcd(a, p) = 1$ 时 $a^{p-1} \equiv 1 \pmod p$。可利用 Montgomery 算术计算椭圆曲线的点乘（见 3.3 节），其中还用到了 2.2.5 节中将介绍的 Montgomery 求逆方法。

2.2.5　求逆

首先回忆一下，在有限域 $\mathbb{F}_p$ 上的非零元素 a 的逆记为 $a^{-1} \bmod p$，或直接简记为 a^{-1}。即在有限域 $\mathbb{F}_p$ 上存在惟一的一个元素 x，使得 $ax \equiv 1 \pmod p$，则元素 x 为 a 的逆 a^{-1}。可以用扩展的整数 Euclidean 算法来求逆。

扩展的整数 Euclidean 算法

令 a 和 b 是非零整数，则 a 和 b 的最大公因子表示为 $\gcd(a, b)$。a 和 b 的最大公因子 $\gcd(a, b)$ 是能同时整除 a 和 b 的最大整数 d。计算 $\gcd(a, b)$ 的算法可根据下面的结论得出。

定理 2.18 令 a 和 b 是正整数，则对于所有的整数 c，都有 $\gcd(a,b)=\gcd(b-ca,a)$。

在计算正整数 a 和 b（$b \geqslant a$）的最大公因子 $\gcd(a,b)$ 的传统 Euclidean 算法中，用 a 除 b，得到满足 $b=qa+r$，$0 \leqslant r<a$ 的商 q 和余数 r。由定理 2.18，$\gcd(a,b)=\gcd(r,a)$。因此确定 $\gcd(a,b)$ 的问题，就简化为确定 $\gcd(r,a)$ 的问题，其中变量 (r,a) 小于原来的变量 (a,b)。这一过程一直持续进行，直到变量中一个为 0 时止，因为 $\gcd(0,d)=d$，所以得到最终结果 $\gcd(a,b)$。这一算法必定终止，因为非负余数序列是严格递减的。这个算法是有效的。设 a 的二进制长度为 k，则算法中除法的次数最多为 $2k$ 次。

Euclidean 算法还可以扩展用于寻找出两个整数 x 和 y，使得 $ax+by=d$，其中 $d=\gcd(a,b)$。算法 2.19 保持下列关系式不变：

$$ax_1+by_1=u,\quad ax_2+by_2=v,\quad u \leqslant v。$$

当 $u=0$ 时算法终止，此时 $v=\gcd(a,b)$，并且 $x=x_2$，$y=y_2$，满足 $ax+by=d$。

算法 2.19 扩展的整数 Euclidean 算法

输入：正整数 a 和 b，且 $a \leqslant b$。

输出：$d=\gcd(a,b)$，满足 $ax+by=d$ 的整数 x 和 y。

1. $u \leftarrow a, v \leftarrow b$。
2. $x_1 \leftarrow 1, y_1 \leftarrow 0, x_2 \leftarrow 0, y_2 \leftarrow 1$。
3. 当 $u \neq 0$ 时，重复执行
 3.1 $q \leftarrow \lfloor v/u \rfloor, r \leftarrow v-qu, x \leftarrow x_2-qx_1, y \leftarrow y_2-qy_1$。
 3.2 $v \leftarrow u, u \leftarrow r, x_2 \leftarrow x_1, x_1 \leftarrow x, y_2 \leftarrow y_1, y_1 \leftarrow y$。
4. $d \leftarrow v, x \leftarrow x_2, y \leftarrow y_2$。
5. 返回 (d,x,y)。

设 p 是一个素数，且 $a \in [1,p-1]$，则 $\gcd(a,p)=1$。若算法 2.19 对于输入 (a,p) 执行，则在步骤 3.1 遇到的最后的非零余数 $r=1$。在步骤 3.2，u, x_1, y_1 被更新，以满足 $ax_1+py_1=u$，而 $u=1$。因此 $ax_1 \equiv 1 \mod p$，而 $a^{-1}=x_1 \pmod p$。注意，确定 x_1 并不需要 y_1 和 y_2。根据这些分析，可导出求有限域 $\mathbb{F}_p$ 上元素逆的算法 2.20。

算法 2.20 利用扩展的 Euclidean 算法求 $\mathbb{F}_p$ 上的逆

输入：素数 p 和 $a \in [1,p-1]$。

输出：$a^{-1} \bmod p$。

1. $u \leftarrow a, v \leftarrow p$。
2. $x_1 \leftarrow 1, x_2 \leftarrow 0$。
3. 当 $u \neq 1$时，重复执行
 3.1　$q \leftarrow \lfloor v/u \rfloor, r \leftarrow v - qu, x \leftarrow x_2 - qx_1$。
 3.2　$v \leftarrow u, u \leftarrow r, x_2 \leftarrow x_1, x_1 \leftarrow x$。
4. 返回（$x_1 \bmod p$）。

二进制逆算法

算法 2.20 的一个缺点是在步骤 3.1 需要复杂的除法运算。这个二进制逆算法用简单的移位操作（除 2）和减法代替了原来复杂的除法。随后我们将给出这个二进制 gcd 算法的一个扩展算法。

在算法 2.21 的步骤 3.1 的每次迭代之前，u 和 v 中最多有一个是奇数，因此在步骤 3.1 和步骤 3.2 的除 2 操作不改变$\gcd(u, v)$的值。在每次迭代中步骤 3.1 和步骤 3.2 执行之后，u 和 v 都变成奇数。并且在步骤 3.3 之后 u 和 v 中恰好有一个成为偶数。因此，步骤 3 的每一次迭代都缩减了 u 或 v 的二进制长度。因此最多迭代 $2k$ 次，其中 k 是 a 或 b 的二进值长度的最大值。

算法 2.21　二进制 gcd 算法

输入：正整数 a 和 b。

输出：$\gcd(a, b)$。

1. $u \leftarrow a, v \leftarrow b, e \leftarrow 1$。
2. 当 u 和 v 都是偶数时，重复执行 $u \leftarrow u/2, v \leftarrow v/2, e \leftarrow 2e$。
3. 当 $u \neq 0$时，重复执行
 3.1　当 u 是偶数时，重复执行 $u \leftarrow u/2$。
 3.2　当 v 是偶数时，重复执行 $v \leftarrow v/2$。
 3.3　若 $u \geqslant v$，则 $u \leftarrow u - v$；否则，$v \leftarrow v - u$。
4. 返回$(e \cdot v)$。

算法 2.22 通过寻找满足 $ax + py = 1$的整数 x 来计算 $a^{-1} \bmod p$。算法保持下列关系式不变：

$$ax_1 + py_1 = u, \quad ax_2 + py_2 = v,$$

其中 y_1 和 y_2 不被计算。当 $u = 1$ 或 $v = 1$ 时算法终止。对于前一种情况，$ax_1 + py_1 = 1$，因

此 $a^{-1}=x_1 \mod p$。对于后一种情况，$ax_2+py_2=1$，因此 $a^{-1}=x_2 \mod p$。

算法 2.22 求 $\mathbb{F}_p$ 上的逆的二进制算法

输入：素数 p 和 $a\in[1,p-1]$。

输出：$a^{-1} \mod p$。

1. $u\leftarrow a, v\leftarrow p$。
2. $x_1\leftarrow 1, x_2\leftarrow 0$。
3. 当 $u\neq 1$ 和 $v\neq 1$ 时，重复执行
 3.1 当 u 是偶数时，重复执行
 $u\leftarrow u/2$。
 若 x_1 是偶数，则 $x_1\leftarrow x_1/2$；否则，$x_1\leftarrow (x_1+p)/2$。
 3.2 当 v 是偶数时，重复执行
 $v\leftarrow v/2$。
 若 x_2 是偶数，则 $x_2\leftarrow x_2/2$；否则，$x_2\leftarrow (x_2+p)/2$。
 3.3 若 $u\geqslant v$，则 $u\leftarrow u-v, x_1\leftarrow x_1-x_2$；否则，$v\leftarrow v-u, x_2\leftarrow x_2-x_1$。
4. 若 $u=1$，则返回（$x_1 \mod p$）；否则，返回（$x_2 \mod p$）。

把求 $\mathbb{F}_p$ 上元素逆的二进制算法的初始条件 $x_1\leftarrow 1$ 修改为 $x_1\leftarrow b$，便可得到计算 $b/a=ba^{-1} \mod p$ 的除法算法。修改后的算法和原来的算法在执行时间上是相同的，因为在求逆算法中少量的迭代后 x_1 的二进制长度就是全长了。2.3 节将详细讨论二进制域的除法算法，其中相对于乘法而言低成本的求逆使得除法变得特别有吸引力。

算法 2.22 可以转化成一个两级求逆的过程，首先对于某个整数 $k\geqslant 0$ 寻找 $a^{-1}2^k \mod p$，然后再求出 a^{-1}。这种变化与算法 2.50 的二进制域求逆方法非常类似，并允许对算法 2.22 中不适用的部分进行优化。下面的 Montgomery 求逆一节中概括介绍了这一基本方法。

Montgomery 求逆

如在 2.2.4 节介绍的那样，Montgomery 方法的基本思想是对于特别选择的 R，用低成本的 $zR^{-1} \mod p$ 运算代替了复杂的 $z \mod p$ 运算。Montgomery 方法可以看做是对表示式 $\tilde{x}=xR \mod p$ 的运算，它特别适用于模幂计算。在计算的开始需要一个辅助变换 $x\mapsto \tilde{x}$，在最后也需要一个辅助变换 $\tilde{x}\mapsto \tilde{x}R^{-1} \mod p=x$，这些辅助变换对于整个计算量来说是不多的。

令 p 是大于 2 的奇整数（可能是合数），定义 $n=\lceil \mathrm{lb}\, p\rceil$。满足 $\gcd(a,p)=1$ 的整数 a 的

Montgomery 求逆是$a^{-1}2^n \bmod p$。算法 2.23 是对算法 2.22 的修改，它用于计算$a^{-1}2^k \bmod p$，对于某个整数$k \in [n, 2n]$。

算法 2.23　$\mathbb{F}_p$上的部分 Montgomery 逆

输入：奇整数$p > 2$，$a \in [1, p-1]$，$n = \lceil \mathrm{lb}\, p \rceil$。

输出：或“无逆”，或(x, k)，其中$n \leqslant k \leqslant 2n$且$x = a^{-1}2^k \bmod p$。

1. $u \leftarrow a, v \leftarrow p, x_1 \leftarrow 1, x_2 \leftarrow 0, k \leftarrow 0$。
2. 当$v > 0$时，重复执行
 2.1　若v是偶数，则$v \leftarrow v/2, x_1 \leftarrow 2x_1$；
 否则，若u是偶数，则$u \leftarrow u/2, x_2 \leftarrow 2x_2$；
 否则，若$v \geqslant u$，则$v \leftarrow (v-u)/2, x_2 \leftarrow x_2 + x_1, x_1 \leftarrow 2x_1$；
 否则，$u \leftarrow (u-v)/2, x_1 \leftarrow x_2 + x_1, x_2 \leftarrow 2x_2$。
 2.2　$k \leftarrow k+1$。
3. 若$u \neq 1$，则返回（“无逆”）。
4. 若$x_1 > p$，则$x_1 \leftarrow x_1 - p$。
5. 返回(x_1, k)。

对于不可逆的a，Montgomery 逆$a^{-1}2^n \bmod p$可以根据输出(x, k)用$k-n$重复去除的方式得到：

$$\text{若 } x \text{ 是偶数，则 } x \leftarrow x/2\text{；否则，} x \leftarrow (x+p)/2\text{。} \tag{2.2}$$

与二进制方法（算法 2.22）相比，为了得到二进制逆，算法 2.23 简单地更新变量x_1和x_2，虽然发生在式(2.2)中的$k-n$的更新更复杂。

注 2.24　（*算法 2.23 实现方法的正确性*）

(i) 除了$\gcd(u, v) = \gcd(a, p)$之外，不变式

$$ax_1 \equiv u2^k \pmod{p}, \quad ax_2 \equiv -v2^k \pmod{p}$$

也成立。若$\gcd(a, p) = 1$，则在步骤 2 的最后一次迭代后$u = 1$并且$x_1 \equiv a^{-1}2^k \pmod{p}$。

(ii) 直至最后一次迭代前，条件

$$p = vx_1 + ux_2, \quad x_1 \geqslant 1, \quad v \geqslant 1, \quad 0 \leqslant u \leqslant a$$

都成立，因此$x_1, v \in [1, p]$。而在最后一次迭代时，$x_1 \leftarrow 2x_1 \leqslant 2p$；若$\gcd(a, p) = 1$，则必须有$x_1 < 2p$且步骤4确保$x_1 < p$。与算法2.22不同，变量$x_1$和$x_2$增长得慢，可能允许算法实现时进行优化。

(iii) 步骤2的每一次迭代都把乘积uv最少约减一半，而和$u+v$最多约减一半。初始时$u+v=a+p$且$uv=ap$，在最后一次迭代前$u=v=1$。因此，$(a+p)/2 \leqslant 2^{k-1} \leqslant ap$，致使$2^{n-2} < 2^{k-1} < 2^{2n}$且$n \leqslant k \leqslant 2n$。

为了获得高效率，通常Montgomery方法选择$R = 2^{Wt} \geqslant 2^n$并采用表示法$\tilde{x} = xR \mod p$。$\tilde{x}$和$\tilde{y}$的Montgomery积$\text{Mont}(\tilde{x}, \tilde{y})$如式(2.1)所定义。式(2.2)的计算可利用Montgomery乘由a来计算产生$a^{-1} \mod p$或$a^{-1}R \mod p$（而不是$a^{-1}2^n \mod p$），或者当算法2.23已采用$\tilde{a}$表示法（而不是a）时计算$a^{-1}R \mod p$。若Montgomery方法采用仿射坐标，则算法2.25还可用于椭圆曲线的点乘计算（见3.3节）。

算法2.25 $\mathbb{F}_p$上的Montgomery逆

输入：奇整数$p > 2$，$n = \lceil \text{lb } p \rceil$，$R^2 \mod p$，$\tilde{a} = aR \mod p$，而$\gcd(a, p) = 1$。

输出：$a^{-1}R \mod p$。

1. 应用算法2.23找出(x, k)，其中$x = \tilde{a}^{-1}2^k \mod p$且$n \leqslant k \leqslant 2n$。
2. 若$k < Wt$，则
 2.1 $x \leftarrow \text{Mont}(x, R^2) = a^{-1}2^k \mod p$。
 2.2 $k \leftarrow k + Wt$（现在$k > Wt$）。
3. $x \leftarrow \text{Mont}(x, R^2) = a^{-1}2^k \mod p$。
4. $x \leftarrow \text{Mont}(x, 2^{2Wt-k}) = a^{-1}R \mod p$。
5. 返回(x)。

采用算法2.22的变体，以$R^2 \mod p$和$\tilde{a}$作为输入，也可计算得到$a^{-1}R \equiv R^2/(aR) \pmod p$。然而算法2.25的计算效率更高，并且Montgomery乘比求逆的成本更低。

同时求逆

域上求逆比乘法要复杂。若需要对几个元素求逆，则用同时求逆的方法需要一次求逆和大约每个元素3次乘法运算。这种方法的基础是$1/x = y(1/xy)$和$1/y = x(1/xy)$。在算法2.26中元素的个数扩大到k。

算法 2.26　同时求逆

输入：素数 p 和域 $\mathbb{F}_p$ 上的非零元素 $a_1, \ldots, a_k$。

输出：域元素 $a_1^{-1}, \ldots, a_k^{-1}$，其中 $a_i a_i^{-1} \equiv 1 \pmod{p}$。

1. $c_1 \leftarrow a_1$。
2. 对于 i 从 2 到 k，重复执行：$c_i \leftarrow c_{i-1} a_i \bmod p$。
3. $u \leftarrow c_k^{-1} \bmod p$。
4. 对于 i 从 k 到 2，重复执行
 4.1　$a_i^{-1} \leftarrow u c_{i-1} \bmod p$。
 4.2　$u \leftarrow u a_i \bmod p$。
5. $a_1^{-1} \leftarrow u$。
6. 返回 $(a_1^{-1}, \ldots, a_k^{-1})$。

对于 k 个元素，算法需要一次求逆、$3(k-1)$ 次乘法以及 k 个元素的临时存储空间。虽然这里给出的算法是针对素域的，但是技术仍适用于其他域。只要求一次逆的成本高于三次乘法，则这种方法就优于分别单独求 k 个逆。

2.26　NIST 素数

美国 FIPS 186-2 标准推荐采用 5 个素域上的椭圆曲线，这 5 个素数模分别是

$$
\begin{aligned}
p_{192} &= 2^{192} - 2^{64} - 1 \\
p_{224} &= 2^{224} - 2^{96} + 1 \\
p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\
p_{521} &= 2^{521} - 1
\end{aligned}
$$

这些素数具有这样的特性：它们都可以表示成少量的 2 的幂的和或差。进而，除了 p_{521} 外，这些素数的上述表达式中 2 的幂都是 32 的倍数。这一特性使得在字长 32 位的机器上约减算法的计算特别快。

例如，考虑 $p = p_{192} = 2^{192} - 2^{64} - 1$。令 c 是一个 $0 \leqslant c < p^2$ 的整数。再令

$$
c = c_5 2^{320} + c_4 2^{256} + c_3 2^{192} + c_2 2^{128} + c_1 2^{64} + c_0 \qquad (2.3)
$$

是 c 的 2^{64} 进制的表示式，其中 $c_i \in [0, 2^{64} - 1]$。我们可以利用以下同余式约减式(2.3)中的

2 的高次幂项：

$$2^{192} \equiv 2^{64} + 1 \pmod{p}$$
$$2^{256} \equiv 2^{128} + 2^{64} \pmod{p}$$
$$2^{320} \equiv 2^{128} + 2^{64} + 1 \pmod{p}。$$

于是我们得到

$$\begin{aligned} c \equiv\ & c_5 2^{128} + c_5 2^{64} + c_5 \\ & + c_4 2^{128} + c_4 2^{64} \\ & + \qquad c_3 2^{64} + c_3 \\ & + c_2 2^{128} + c_1 2^{64} + c_0 \pmod{p}。 \end{aligned}$$

因此，通过 4 次 192 位的整数 $c_5 2^{128} + c_5 2^{64} + c_5$, $c_4 2^{128} + c_4 2^{64}$, $c_3 2^{64} + c_3$ 和 $c_2 2^{128} + c_1 2^{64} + c_0$相加，并重复地减 p，直到结果小于 p 为止，便可得到 $c \bmod p$。

算法 2.27 模 $p_{192} = 2^{192} - 2^{64} - 1$的快速约减

输入：$c = (c_5, c_4, c_3, c_2, c_1, c_0)$为 2^{64} 进制整数，$0 \leqslant c < p_{192}^2$。

输出：$c \bmod p_{192}$。

1. 定义 192 位整数：

 $s_1 = (c_2, c_1, c_0)$, $s_2 = (0, c_3, c_3)$,

 $s_3 = (c_4, c_4, 0)$, $s_4 = (c_5, c_5, c_5)$。

2. 返回 $(s_1 + s_2 + s_3 + s_4 \bmod p_{192})$。

算法 2.28 模 $p_{224} = 2^{224} - 2^{96} + 1$的快速约减

输入：$c = (c_{13}, \ldots, c_2, c_1, c_0)$为 2^{32} 进制整数，$0 \leqslant c < p_{224}^2$。

输出：$c \bmod p_{224}$。

1. 定义 224 位整数：

 $s_1 = (c_6, c_5, c_4, c_3, c_2, c_1, c_0)$, $s_2 = (c_{10}, c_9, c_8, c_7, 0, 0, 0)$,

 $s_3 = (0, c_{13}, c_{12}, c_{11}, 0, 0, 0)$, $s_4 = (c_{13}, c_{12}, c_{11}, c_{10}, c_9, c_8, c_7)$,

 $s_5 = (0, 0, 0, 0, c_{13}, c_{12}, c_{11})$。

2. 返回$(s_1 + s_2 + s_3 - s_4 - s_5 \bmod p_{224})$。

算法 2.29　模 $p_{256}=2^{256}-2^{224}+2^{192}+2^{96}-1$ 的快速约减

输入：$c=(c_{15},\ldots,c_2,c_1,c_0)$ 为 2^{32} 进制整数，$0\leqslant c<p_{256}^2$。

输出：$c \bmod p_{256}$。

1. 定义 256 位整数：

$s_1=(c_7,c_6,c_5,c_4,c_3,c_2,c_1,c_0)$，

$s_2=(c_{15},c_{14},c_{13},c_{12},c_{11},0,0,0)$，

$s_3=(0,c_{15},c_{14},c_{13},c_{12},0,0,0)$，

$s_4=(c_{15},c_{14},0,0,0,c_{10},c_9,c_8)$，

$s_5=(c_8,c_{13},c_{15},c_{14},c_{13},c_{11},c_{10},c_9)$，

$s_6=(c_{10},c_8,0,0,0,c_{13},c_{12},c_{11})$，

$s_7=(c_{11},c_9,0,0,c_{15},c_{14},c_{13},c_{12})$，

$s_8=(c_{12},0,c_{10},c_9,c_8,c_{15},c_{14},c_{13})$，

$s_9=(c_{13},0,c_{11},c_{10},c_9,0,c_{15},c_{14})$。

2. 返回 $(s_1+2s_2+2s_3+s_4+s_5-s_6-s_7-s_8-s_9 \bmod p_{256})$。

算法 2.30　模 $p_{384}=2^{384}-2^{128}-2^{96}+2^{32}-1$ 的快速约减

输入：$c=(c_{23},\ldots,c_2,c_1,c_0)$ 为 2^{32} 进制整数，$0\leqslant c<p_{384}^2$。

输出：$c \bmod p_{384}$。

1. 定义 384 位整数：

$s_1=(c_{11},c_{10},c_9,c_8,c_7,c_6,c_5,c_4,c_3,c_2,c_1,c_0)$，

$s_2=(0,0,0,0,0,c_{23},c_{22},c_{21},0,0,0,0)$，

$s_3=(c_{23},c_{22},c_{21},c_{20},c_{19},c_{18},c_{17},c_{16},c_{15},c_{14},c_{13},c_{12})$，

$s_4=(c_{20},c_{19},c_{18},c_{17},c_{16},c_{15},c_{14},c_{13},c_{12},c_{23},c_{22},c_{21})$，

$s_5=(c_{19},c_{18},c_{17},c_{16},c_{15},c_{14},c_{13},c_{12},c_{20},0,c_{23},0)$，

$s_6=(0,0,0,0,c_{23},c_{22},c_{21},c_{20},0,0,0,0)$，

$s_7=(0,0,0,0,0,0,c_{23},c_{22},c_{21},0,0,c_{20})$，

$s_8=(c_{22},c_{21},c_{20},c_{19},c_{18},c_{17},c_{16},c_{15},c_{14},c_{13},c_{12},c_{23})$，

$s_9=(0,0,0,0,0,0,0,c_{23},c_{22},c_{21},c_{20},0)$，

$s_{10}=(0,0,0,0,0,0,0,c_{23},c_{23},0,0,0)$。

2. 返回 $(s_1+2s_2+s_3+s_4+s_5+s_6+s_7-s_8-s_9-s_{10} \bmod p_{384})$。

算法 2.31 模 $p_{521}=2^{521}-1$的快速约减

输入：$c=(c_{1041},\ldots,c_2,c_1,c_0)$为二进制整数，$0\leqslant c<p_{521}^2$。

输出：$c \bmod p_{521}$。

1. 定义 521 位整数：

 $s_1=(c_{1041},\ldots,c_{523},c_{522},c_{521})$，

 $s_2=(c_{520},\ldots,c_2,c_1,c_0)$。

2. 返回$(s_1+s_2 \bmod p_{521})$。

2.3 二进制域算术

本节给出的算法都是适合软件实现的二进制域算术运算的。第 5 章介绍了利用某些处理器中的单指令多数据(SIMD)寄存器实现域运算（见 5.1.3 节）以及用硬件实现域运算的设计考虑（见 5.2.2 节）。5.1.5 节中介绍了域运算的定时问题。

我们假设实现平台是字长 W 位的系统，其中 W 是 8 的整数倍。一个字 U 的 W 位，分别以 0 到 W–1 标记，其中第 0 位为最高有效位。下面的标准符号用于表示字 U 和字 V 的运算：

$U\oplus V$	按位异或
$U\,\&\,V$	按位与
$U\gg i$	将 U 右移 i 位，左边 i 位补 0
$U\ll i$	将 U 左移 i 位，右边 i 位补 0

设 $f(z)$ 是一个 m 次的二进制既约多项式，记 $f(z)=z^m+r(z)$，则域$\mathbb{F}_{2^m}$的元素是次数最多为 $m-1$ 的二进制多项式。域元素的加法就是二进制多项式的加法。域元素的乘法就是二进制多项式相乘并取模 $f(z)$。一个域元素$a(z)=a_{m-1}z^{m-1}+\cdots+a_2z^2+a_1z+a_0$与一个 m 维向量$a=(a_{m-1},\ldots,a_2,a_1,a_0)$相对应。令$t=\lceil m/W\rceil$，$s=Wt-m$。在软件实现中可以用 t 个 W 字的一个数组 $A=(A[t-1],\ldots,A[2],A[1],A[0])$来存储 a，最高有效位 a_0 存储到 $A[0]$ 中，并且$A[t-1]$的最左 s 位没有用（总是设置为 0）。

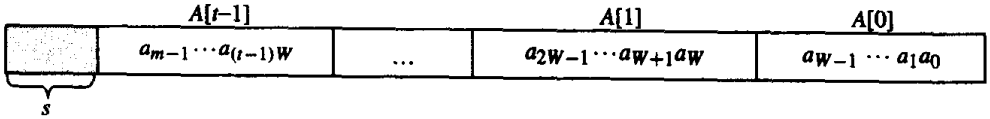


图 2.4 用 W 位的字表示域$\mathbb{F}_{2^m}$的元素 a。$A[t-1]$的最高$s=tW-m$位未用

2.3.1　加法

域元素的加法是按位进行的，从而总共只需要 t 个字运算。

算法 2.32　域 $\mathbb{F}_{2^m}$ 上的加法

输入：次数低于 m 的二进制多项式 $a(z)$，$b(z)$。

输出：$c(z)=a(z)+b(z)$。

1. 对于 i 从 0 到 $t-1$，重复执行

 1.1　$C[i]\leftarrow A[i]\oplus B[i]$。

2. 返回(c)。

2.3.2　乘法

用移位加的方法实现域的乘法（算法 2.33）建立在下面的基础之上：

$$a(z)\cdot b(z)=a_{m-1}z^{m-1}b(z)+\cdots+a_2z^2b(z)+a_1zb(z)+a_0b(z)。$$

在算法中对 i 实施迭代，计算 $z^ib(z) \bmod f(z)$，并且若 $a_i=1$，则把结果加到累加器 c 上。若 $b(z)=b_{m-1}z^{m-1}+\cdots+b_2z^2+b_1z+b_0$，则

$$\begin{aligned} b(z)\cdot z &= b_{m-1}z^m+b_{m-2}z^{m-1}+\cdots+b_2z^3+b_1z^2+b_0z \\ &\equiv b_{m-1}r(z)+(b_{m-2}z^{m-1}+\cdots+b_2z^3+b_1z^2+b_0z) \pmod{f(z)}。 \end{aligned}$$

这样，$b(z)\cdot z \bmod f(z)$就可以通过对$b(z)$的向量表示进行左移来实现，边移位边判断，若高阶位b_{m-1}为 1，则把$r(z)$加到$b(z)$上。

算法 2.33　从右向左移位加计算域 $\mathbb{F}_{2^m}$ 上的乘法

输入：次数低于 m 的二进制多项式 $a(z)$，$b(z)$。

输出：$c(z)=a(z)\cdot b(z) \bmod f(z)$。

1. 若$a_0=1$，则$c\leftarrow b$；否则，$c\leftarrow 0$。

2. 对于 i 从 1 到 $m-1$，重复执行

 2.1　$b\leftarrow b\cdot z \bmod f(z)$。

 2.2　若 $a_i=1$，则$c\leftarrow c+b$。

3. 返回(c)。

算法 2.33 是非常适合硬件实现的，其中向量的一次移位可用一个时钟周期完成，但大

量的字移位使得它不适合软件实现。下面我们将讨论快速实现域乘法的方法，它首先进行域元素多项式的乘（见 2.3.3 节和 2.3.4 节），然后对模$f(z)$约减（见 2.3.5 节）。

2.3.3 多项式乘法

从右向左的 comb 多项式乘法（算法 2.34）建立在下面的观察基础之上：若对于某些 k，$k \in [0, W-1]$，$b(z) \cdot z^k$已经计算出来，则通过在$b(z) \cdot z^k$的表示向量的右边附加 j 个 0 字，便很容易地获得$b(z) \cdot z^{Wj+k}$。算法 2.34 从右向左处理 A 字的各位，如图 2.5 所示。图 2.5 中的参数 $m = 163$，$W = 32$。我们采用下列符号：若$C = (C[n], \ldots, C[2], C[1], C[0])$是一个数组，则$C\{j\}$表示截短数组$(C[n], \ldots, C[j+1], C[j])$。

A[0]	a_{31}	...	a_2	a_1	a_0
A[1]	a_{63}	...	a_{34}	a_{33}	a_{32}
A[2]	a_{95}	...	a_{66}	a_{65}	a_{64}
A[3]	a_{127}	...	a_{98}	a_{97}	a_{96}
A[4]	a_{159}		a_{130}	a_{129}	a_{128}
A[5]			a_{162}	a_{161}	a_{160}

图 2.5 从右向左的 comb 方法（算法 2.34）按从右向左的方向处理二维数组的列，处理每列的位按从上向下的方向进行。图中的参数是 $W = 32$，$m = 163$

算法 2.34 从右向左的 comb 多项式乘法

输入：次数低于 m 的二进制多项式$a(z)$，$b(z)$。

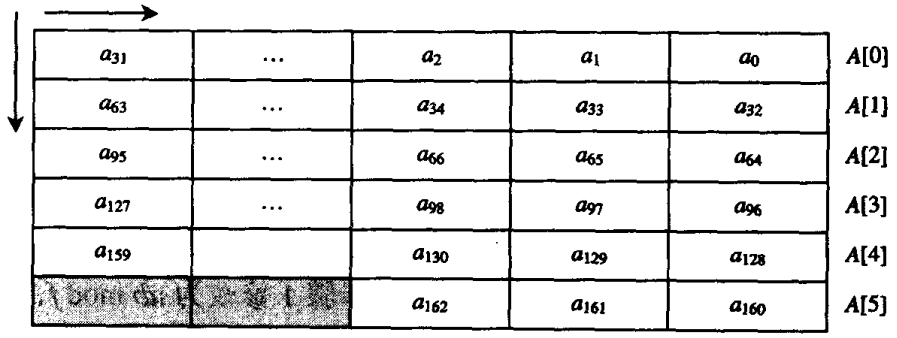
输出：$c(z) = a(z) \cdot b(z)$。

1. $C \leftarrow 0$。
2. 对于 k 从 0 到$W-1$，重复执行
 2.1 对于 j 从 0 到 $t-1$，重复执行
 若$A[j]$的第 k 位是 1，则 $B + C\{j\}$。
 2.2 若 $k \neq (W-1)$，则$B \leftarrow B \cdot z$。
3. 返回(C)。

从右向左的 comb 多项式乘法按从左向右的方向处理 a 的位，如下所示：

$$a(z) \cdot b(z) = \Big(\cdots\big((a_{m-1}b(z)z + a_{m-2}b(z))z + a_{m-3}b(z)\big)z + \cdots + a_1 b(z)\Big)z + a_0 b(z)。$$

算法 2.35 是对这一方法的改进，其中字 A 的位处理按从左向右的方向进行。图 2.6 显示了这种改进，图中参数 $m = 163$，$W = 32$。



a_{31}	…	a_2	a_1	a_0	A[0]
a_{63}	…	a_{34}	a_{33}	a_{32}	A[1]
a_{95}	…	a_{66}	a_{65}	a_{64}	A[2]
a_{127}	…	a_{98}	a_{97}	a_{96}	A[3]
a_{159}		a_{130}	a_{129}	a_{128}	A[4]
		a_{162}	a_{161}	a_{160}	A[5]

图 2.6　从左到向右的 comb 方法（算法 2.35）按从左向右的方向处理二维数组的列，处理每列的位按从上向下的方向进行。图中的参数是 $W=32$，$m=163$

算法 2.35　从左向右的 comb 多项式乘法

输入：次数低于 m 的二进制多项式$a(z)$，$b(z)$。

输出：$c(z)=a(z)\cdot b(z)$。

1. $C\leftarrow 0$。
2. 对于 k 从 $W-1$ 到 0，重复执行
 2.1 对于 j 从 0 到 $t-1$，重复执行
 若$A[j]$的第 k 位是 1，则 $B+C\{j\}$。
 2.2 若 $k\neq 0$，则$C\leftarrow C\cdot z$。
3. 返回(C)。

算法 2.34 和算法 2.35 都比算法 2.33 快，因为它们只有少量的向量移位（用 z 乘）。算法 2.34 比算法 2.35 还要快，因为算法 2.34 的向量移位涉及 t 个字的数组 B（它可以增长到 $t+1$ 个字），而算法 2.35 的向量移位涉及 $2t$ 个字的数组 C。

通过对于所有次数低于w的多项式$u(z)$，事先计算出$u(z)\cdot b(z)$，并存储乘积，处理时查表一次计算出$A[j]$的w位，就可使算法 2.35 大大加速。这种改进的方法示于算法 2.36 中，其中 a 的位处理方向如图 2.7 所示。图 2.7 中的参数是 $M=163$，$W=32$，$w=4$。

算法 2.36　从左向右的 comb 多项式乘法（窗口宽度 w）

输入：次数低于 m 的二进制多项式$a(z)$，$b(z)$。

输出：$c(z)=a(z)\cdot b(z)$。

1. 计算$B_u=u(z)\cdot b(z)$，对于所有次数低于w的多项式$u(z)$。
2. $C\leftarrow 0$。
3. 对于 k 从$(W/w)-1$到 0，重复执行

3.1 对于j从0到$t-1$，重复执行

令$u=(u_{w-1},\ldots,u_1,u_0)$，其中$u_i$是$A[j]$的第$(wk+i)$位，$B_u+C\{j\}$。

3.2 若$k\neq 0$，则$C\leftarrow C\cdot z^w$。

4. 返回(C)。

算法2.36只完成多项式的乘法，而域乘法的模约减运算要另外进行。在某些情况下，把约减多项式f作为算法的输入是有益的。此时把步骤1修改为$ub \bmod f$，这样做还可以优化步骤3。

注2.37 （*算法2.36的加强*）与处理器的特性相关，算法2.36的一种有益的修改方案改变了移位加和查表的顺序。预计算对l个表进行；为简单起见，我们假设$l \mid w$。对于所有次数低于w/l的多项式造表，表i由值$B_{v,i}=v(z)z^{iw/l}b(z)$组成，$0\leqslant i<l$。算法2.36的步骤3.1改成计算$B_u=\sum_{i=0}^{l-1}B_{u^i,i}$，其中$u=(u_{w-1},\ldots,u_0)=(u^{l-1},\ldots,u^0)$，且$u^i$有$w/l$位。作为一个例子，$w=4$的算法2.36有16个元素要预计算。对于参数$w=8$，$l=4$，改进算法的预计算量与原算法相同（4个表，每个表4个点）。与原来的算法相比，改进的算法在步骤3只有少量的迭代（在步骤3.2只有少量的移位）。然而步骤3.1是复杂的。

Comb方法是由López和Dahab提出的，并且是以Lim和Lee提出的可修改用于二进制域的指数方法为基础的。3.3.2节将更详细地讨论Lim和Lee的方法在椭圆曲线点乘中的应用，参见注3.47。

Karatsuba-Ofman乘法

2.2.2节介绍的由Karatsuba-Ofman提出的分而治之方法，可以直接推广到多项式情况。例如，

$$\begin{aligned}a(z)b(z)&=(A_1z^l+A_0)(B_1z^l+B_0)\\&=A_1B_1z^{2l}+[(A_1+A_0)(B_1+B_0)+A_1B_1+A_0B_0]z^l+A_0B_0\ ,\end{aligned}$$

其中$l=\lceil m/2\rceil$，A_0，A_1，B_0，B_1是z中次数低于l的二进制多项式。利用查表或其他方法，可以在某一界限范围内重复进行处理。按低于算法2.36的这种策略实施，对于实际所需的m值就够用了。

注2.38 （*多项式乘法的实现*）假如硬件特性相当精确，则对于椭圆曲线方法所感兴趣的二进制域，实际上算法2.36是最快的算法之一。不同的C编译的代码在性能上可能有很大的不同，而且编译可能对算法描述的准确性敏感。

Sun 公司的微系统实验室(SML)于 2002 年开发的 OpenSSL 是一种实际应用中的折中选择。OpenSSL 被广泛应用于为 Apache Web 服务器和 OpenSSH 安全通信工具提供密码服务。SML 的贡献在本书中可以看到。OpenSSL 是公开的和合作的结果，Sun 的代码起了很大作用。

为了保持代码规模较小，SML 开发了一种通用多项式乘法方法。Karatsuba-Ofman 的方法被用到，但仅仅在 2 字的量的乘法中用到，而不是递归应用。在最低级的 1 个字的量的乘法中应用了简化的算法 2.36（$w=2$, $w=3$，$w=4$，分别在 16 位、32 位和 64 位平台上）。和预计一样，结果比算法 2.36 最快的版本要慢得多。我们对 Sun SPARC 和 P6 系列硬件进行了测试，Karatsuba-Ofman 方法的执行比算法 2.36（2 字情况）慢。然而，若相同的代码被用于交叉平台和交叉编译，则 SML 的贡献可能是 OpenSSL 的较好折中。

2.3.4　多项式平方

因为二进制多项式的平方是一种线性操作，所以它比任意两个多项式相乘快得多。即若 $a(z)=a_{m-1}z^{m-1}+\cdots+a_2z^2+a_1z+a_0$，则

$$a(z)^2=a_{m-1}z^{2m-2}+\cdots+a_2z^4+a_1z^2+a_0。$$

$a(z)^2$的二进制表示可以通过以下方式获得：往$a(z)$的二进制表示中相邻的位之间插入一个 0，如图 2.8 所示。为了便于处理，用一个 512 字节的表 T 进行预计算，把 8 位的多项式变换扩充为 16 位的对应多项式。算法 2.39 给出了 $W=32$ 的这种处理过程。

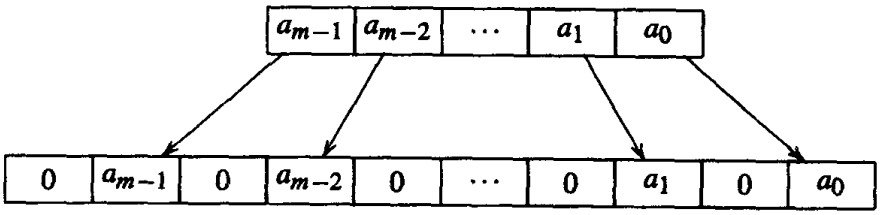


图 2.8　二进制多项式$a(z)=a_{m-1}z^{m-1}+\cdots+a_2z^2+a_1z+a_0$的平方

算法 2.39　多项式平方（字长 $W=32$）

输入：次数低于 m 的二进制多项式$a(z)$。

输出：$c(z)=a(z)^2$。

1. 预计算。对于每个字节 $d=(d_7,\ldots,d_1,d_0)$，产生 16 位的量 $T(d)=(0,d_7,\ldots,0,d_1,0,d_0)$。
2. 对于 i 从 0 到 $t-1$，重复执行

 2.1 令 $A[i]=(u_3,u_2,u_1,u_0)$，其中 u_j是一个字节。

2.2 $C[2i] \leftarrow (T(u_1), T(u_0))$, $C[2i+1] \leftarrow (T(u_3), T(u_2))$。

3. 返回(c)。

2.3.5 约减

我们现在讨论对二进制多项式$c(z)$进行约减的技术，一种是通过对两个次数低于m的二进制多项式相乘，另一种是通过一个次数低于m的二进制多项式平方。$c(z)$的次数最多为$2m-2$。

任意约减多项式

回忆$f(z) = z^m + r(z)$，其中$r(z)$是一个次数最多为$m-1$的二进制多项式。算法 2.40 对$c(z)$模$f(z)$约减，一次处理一位，从最低位开始。可以这样处理的理由如下：

$$\begin{aligned} c(z) &= c_{2m-2}z^{2m-2} + \cdots + c_m z^m + c_{m-1}z^{m-1} + \cdots + c_1 z + c_0 \\ &\equiv (c_{2m-2}z^{m-2} + \cdots + c_m)r(z) + c_{m-1}z^{m-1} + \cdots + c_1 z + c_0 \pmod{f(z)}。 \end{aligned}$$

通过预计算多项式$z^k r(z)$，$0 \leqslant k \leqslant W-1$，约减运算被加速。若多项式$r(z)$的次数很低，或$f(z)$是一个三项式，则所需要的存储空间就很小，而且在步骤 2.1 中涉及$z^k r(z)$的加法的速度就很快。用到下列符号：若$C = (C[n], \ldots, C[2], C[1], C[0])$是一个数组，则$C\{j\}$表示一个截短的数组$(C[n], \ldots, C[j+1], C[j])$。

算法 2.40 模约减（一次一位）

输入：次数最高为$2m-2$的二进制多项式$c(z)$。

输出：$c(z) \bmod f(z)$。

1. 预计算。计算$u_k(z) = z^k r(z)$，$0 \leqslant k \leqslant W-1$。
2. 对于i从$2m-2$到m，重复执行

 2.1 若$c_i = 1$，则

 令$j = \lfloor (i-m)/W \rfloor$，$k = (i-m) - Wj$。

 $u_k(z) + C\{j\}$。
3. 返回$(C[t-1], \ldots, C[1], C[0])$。

若$f(z)$是一个三项式，或是一个中间项互相靠近的五项式（pentanimial），则约减运算$c(z) \bmod f(z)$就是非常有效的，一次处理一个字。例如，设$m = 163$，$W = 32$（因此$t = 6$），考虑$c(z)$模$f(z) = z^{163} + z^7 + z^6 + z^3 + 1$的字$C[9]$的约减。字$C[9]$表示多项式

$c_{319}z^{319}+\cdots+c_{289}z^{289}+c_{288}z^{288}$。我们有

$$\begin{aligned} z^{288} &\equiv z^{132}+z^{131}+z^{128}+z^{125} \pmod{f(z)},\\ z^{289} &\equiv z^{133}+z^{132}+z^{129}+z^{126} \pmod{f(z)},\\ &\vdots\\ z^{319} &\equiv z^{163}+z^{162}+z^{159}+z^{156} \pmod{f(z)}。\end{aligned}$$

考虑上述同余式右边的四列，我们发现$C[9]$的约减可以通过把$C[9]$四次加到 C 上来实现。而$C[9]$的最高位四次分别加到 C 的 132，131，128 和 125 位上。这一过程如图 2.9 所示。

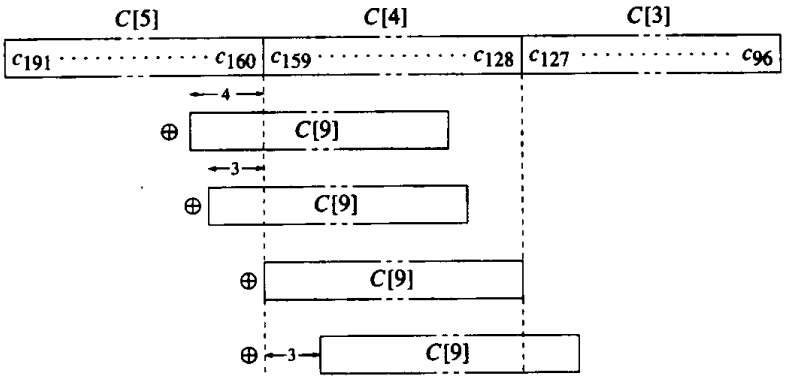


图 2.9　约减 32 位字$C[9]$模 $f(z)=z^{163}+z^7+z^6+z^3+1$

NIST 约减多项式

下面我们将给出一种快速模约减方法，下列约减多项式是 NIST 在 FIPS 186-2 中推荐的：

$$\begin{aligned} f(z) &= z^{163}+z^7+z^6+z^3+1\\ f(z) &= z^{233}+z^{74}+1\\ f(z) &= z^{283}+z^{12}+z^7+z^5+1\\ f(z) &= z^{409}+z^{87}+1\\ f(z) &= z^{571}+z^{10}+z^5+z^2+1 \end{aligned}$$

下面的这些算法基于与图 2.9 类似的思想，并假设字宽 $W=32$。它们比算法 2.40 快得多，并且不需要很多存储空间。

算法 2.41　模 $f(z)=z^{163}+z^7+z^6+z^3+1$ 的快速约减（$W=32$）

输入：次数最高为 324 的二进制多项式 $c(z)$。

输出：$c(z) \bmod f(z)$。

1. 对于 i 从 10 到 6，重复执行　　{约减 $C[i]z^{32i} \bmod f(z)$}

1.1 $T \leftarrow C[i]$。

1.2 $C[i-6] \leftarrow C[i-6] \oplus (T \ll 29)$。

1.3 $C[i-5] \leftarrow C[i-5] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 3)$。

1.4 $C[i-4] \leftarrow C[i-4] \oplus (T \gg 28) \oplus (T \gg 29)$。

2. $T \leftarrow C[5] \gg 3$。 {提取$C[5]$的3~31位}

3. $C[0] \leftarrow C[0] \oplus (T \ll 7) \oplus (T \ll 6) \oplus (T \ll 3) \oplus T$。

4. $C[1] \leftarrow C[1] \oplus (T \gg 25) \oplus (T \gg 26)$。

5. $C[5] \leftarrow C[5] \& 0x7$。 {清除$C[5]$的要约减的位}

6. 返回$(C[5], C[4], C[3], C[2], C[1], C[0])$。

算法 2.42 模 $f(z) = z^{233} + z^{74} + 1$ 的快速约减（$W = 32$）

输入：次数最高为464的二进制多项式 $c(z)$。

输出：$c(z) \bmod f(z)$。

1. 对于 i 从15到8，重复执行 {约减$C[i]z^{32i} \bmod f(z)$}

1.1 $T \leftarrow C[i]$。

1.2 $C[i-8] \leftarrow C[i-8] \oplus (T \ll 23)$。

1.3 $C[i-7] \leftarrow C[i-7] \oplus (T \gg 9)$。

1.4 $C[i-5] \leftarrow C[i-5] \oplus (T \ll 1)$。

1.5 $C[i-4] \leftarrow C[i-4] \oplus (T \gg 31)$。

2. $T \leftarrow C[7] \gg 9$。 {提取$C[7]$的9~31位}

3. $C[0] \leftarrow C[0] \oplus T$。

4. $C[2] \leftarrow C[2] \oplus (T \ll 10)$。

5. $C[3] \leftarrow C[3] \oplus (T \gg 22)$。

6. $C[7] \leftarrow C[7] \& 0x1FF$。 {清除$C[7]$的要约减的位}

7. 返回$(C[7], C[6], C[5], C[4], C[3], C[2], C[1], C[0])$。

算法 2.43 模 $f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$ 的快速约减（$W = 32$）

输入：次数最高为564的二进制多项式 $c(z)$。

输出：$c(z) \bmod f(z)$。

1. 对于 i 从17到9，重复执行 {约减$C[i]z^{32i} \bmod f(z)$}

1.1 $T \leftarrow C[i]$。

1.2　$C[i-9] \leftarrow C[i-9] \oplus (T \ll 5) \oplus (T \ll 10) \oplus (T \ll 12) \oplus (T \ll 17)$。

1.3　$C[i-8] \leftarrow C[i-8] \oplus (T \gg 27) \oplus (T \gg 22) \oplus (T \gg 20) \oplus (T \gg 15)$。

2. $T \leftarrow C[8] \gg 27$。　　{提取$C[8]$的 27~31 位}

3. $C[0] \leftarrow C[0] \oplus T \oplus (T \ll 5) \oplus (T \ll 7) \oplus (T \ll 12)$。

4. $C[8] \leftarrow C[8] \& 0x7FFFFFF$。　　{清除$C[8]$的要约减的位}

5. 返回$(C[8], C[7], C[6], C[5], C[4], C[3], C[2], C[1], C[0])$。

算法 2.44　模 $f(z) = z^{409} + z^{87} + 1$ 的快速约减（$W = 32$）

输入：次数最高为 816 的二进制多项式 $c(z)$。

输出：$c(z) \bmod f(z)$。

1. 对于 i 从 25 到 13，重复执行　　{约减$C[i]z^{32i} \bmod f(z)$}

1.1　$T \leftarrow C[i]$。

1.2　$C[i-13] \leftarrow C[i-13] \oplus (T \ll 7)$。

1.3　$C[i-12] \leftarrow C[i-12] \oplus (T \gg 25)$。

1.4　$C[i-11] \leftarrow C[i-11] \oplus (T \ll 30)$。

1.5　$C[i-10] \leftarrow C[i-10] \oplus (T \gg 2)$。

2. $T \leftarrow C[12] \gg 25$。　　{提取$C[12]$的 25~31 位}

3. $C[0] \leftarrow C[0] \oplus T$。

4. $C[2] \leftarrow C[2] \oplus (T \ll 23)$。

5. $C[12] \leftarrow C[12] \& 0x1FFFFFF$。　　{清除$C[12]$的要约减的位}

6. 返回$(C[12], C[11], \ldots, C[1], C[0])$。

算法 2.45　模 $f(z) = z^{571} + z^{10} + z^5 + z^2 + 1$ 的快速约减（$W = 32$）

输入：次数最高为 1140 的二进制多项式 $c(z)$。

输出：$c(z) \bmod f(z)$。

1. 对于 i 从 35 到 18，重复执行　　{约减$C[i]z^{32i} \bmod f(z)$}

1.1　$T \leftarrow C[i]$。

1.2　$C[i-18] \leftarrow C[i-18] \oplus (T \ll 5) \oplus (T \ll 7) \oplus (T \ll 10) \oplus (T \ll 15)$。

1.3　$C[i-17] \leftarrow C[i-17] \oplus (T \gg 27) \oplus (T \gg 25) \oplus (T \gg 22) \oplus (T \gg 17)$。

2. $T \leftarrow C[17] \gg 27$。　　{提取$C[17]$的 27~31 位}

3. $C[0] \leftarrow C[0] \oplus T \oplus (T \ll 2) \oplus (T \ll 5) \oplus (T \ll 10)$。

4. $C[17] \leftarrow C[17]$ & 0x7FFFFFFF。 {清除$C[17]$的要约减的位}
5. 返回$(C[17], C[16], \ldots, C[1], C[0])$。

2.3.6 求逆和除法

这里我们简单地用 a 来表示二进制多项式 $a(z)$。回忆一下域$\mathbb{F}_{2^m}$的非零元素 a 的逆是域$\mathbb{F}_{2^m}$的惟一的一个元素 g，在域$\mathbb{F}_{2^m}$上满足 $ag = 1$，即满足 $ag \equiv 1 \pmod f$。这个逆元素用符号 $a^{-1} \bmod f$ 表示，若约减多项式不会产生混淆，则可直接表示为 a^{-1}。用多项式的扩展 Euclidean 算法可快速求出逆元素。

多项式的扩展 Euclidean 算法

令 a 和 b 是两个不全为零的二进制多项式。a 和 b 的最大公因式表示为 $\gcd(a, b)$，它是能同时整除 a 和 b 的次数最高的二进制多项式 d。计算 $\gcd(a, b)$的有效算法利用了与定理 2.18 类似的如下定理。

定理 2.46 令 a 和 b 是两个二进制多项式，则对于所有的二进制多项式 c 都有 $\gcd(a, b) = \gcd(b - ca, a)$。

在传统的计算二进制多项式 a 和 b（$\deg(b) \geqslant \deg(a)$）的最大公因式的 Euclidean 算法中，用 a 除 b 得到商 q 和余数 r，满足 $b = qa + r$，且 $\deg(r) < \deg(a)$。根据定理 2.46，$\gcd(a, b) = \gcd(r, a)$，因此确定 $\gcd(a, b)$的问题就简化为计算 $\gcd(r, a)$的问题，其中变量 (r, a)的次数低于原来的变量(a, b)的次数。这一处理过程一致持续到其中一个变量为零为止。这时可立即得到最大公因式，因为 $\gcd(0, d) = d$。算法一定会终止，因为余数的次数是严格递减序列。而且算法是高效的，因为算法的执行最多需要 k 次长除，其中 $k = \deg(a)$。

而在 Euclidean 算法的一种变体中，每一次长除只要一步就能完成。即若 $\deg(b) \geqslant \deg(a)$，且 $j = \deg(b) - \deg(a)$，则一次计算 $r = b + z^j a$。根据定理 2.46，$\gcd(a, b) = \gcd(r, a)$。这一处理过程重复到余数为零为止。因为 $\deg(r) < \deg(b)$，所以除的次数最多为 $2k$，$k = \max\{\deg(a), \deg(b)\}$。

Euclidean 算法还可以扩展用来寻找多项式 g 和 h，使得 $ag + bh = d$，$d = \gcd(a,b)$。算法 2.47 保持下列等式：

$$ag_1 + bh_1 = u$$
$$ag_2 + bh_2 = v。$$

当 $u = 0$ 时算法终止，那时 $v = \gcd(a, b)$，并且 $ag_2 + bh_2 = d$。

算法 2.47　二进制多项式的扩展 Euclidean 算法

输入：非零二进制多项式 a 和 b，且 $\deg(a) \leqslant \deg(b)$。

输出：$d=\gcd(a, b)$，二进制多项式 g 和 h，其中 $ag+bh=d$。

1. $u \leftarrow a, v \leftarrow b$。
2. $g_1 \leftarrow 1, g_2 \leftarrow 0, h_1 \leftarrow 0, h_2 \leftarrow 1$。
3. 当 $u \neq 0$ 时，重复执行
 3.1 $j \leftarrow \deg(u)-\deg(v)$。
 3.2 若 $j<0$，则 $u \leftrightarrow v, g_1 \leftrightarrow g_2, h_1 \leftrightarrow h_2, j \leftarrow -j$。
 3.3 $u=u+z^j v$。
 3.4 $g_1 \leftarrow g_1+z^j g_2, h_1 \leftarrow h_1+z^j h_2$。
4. $d \leftarrow v, g \leftarrow g_2, h \leftarrow h_2$。
5. 返回(d, g, h)。

现在假设 f 是一个次数为 m 的既约二进制多项式，a 是次数最高为 $m-1$ 的非零多项式，显然 $\gcd(a, f)=1$。若算法 2.47 以 f 和 a 为输入进行计算，则在步骤 3.3 就遇到 $u=1$。此后在步骤 3.4 中多项式 g_1 和 h_1 满足 $ag_1+fh_1=1$。因此，$ag_1 \equiv 1 \pmod f$，于是 $a^{-1}=g_1$。注意，对于确定 g_1 的计算，h_1 和 h_2 是不需要的。根据这些分析，得出用于求 $\mathbb{F}_{2^m}$ 上元素逆的算法 2.48。

算法 2.48　用扩展的 Euclidean 算法求 $\mathbb{F}_{2^m}$ 上的逆

输入：次数最高为 $m-1$ 的非零二进制多项式 a。

输出：$a^{-1} \bmod f$。

1. $u \leftarrow a, v \leftarrow f$。
2. $g_1 \leftarrow 1, g_2 \leftarrow 0$。
3. 当 $u \neq 1$ 时，重复执行
 3.1 $j \leftarrow \deg(u)-\deg(v)$。
 3.2 若 $j<0$，则 $u \leftrightarrow v, g_1 \leftrightarrow g_2, j \leftarrow -j$。
 3.3 $u \leftarrow u+z^j v$。
 3.4 $g_1 \leftarrow g_1+z^j g_2$。
4. 返回(g_1)。

求二进制逆的算法

算法 2.49 是求域 $\mathbb{F}_p$ 上元素逆的算法（算法 2.22）的多项式模拟。在算法 2.48 中 u 和 v

的位被清除的顺序是从左到右（从高次项到低次项），而在算法 2.49 中 u 和 v 的位被清除的顺序却是从右到左。

算法 2.49 求$\mathbb{F}_{2^m}$上逆的二进制算法

输入：次数最多为 $m-1$ 的非零二进制多项式 a。

输出：$a^{-1} \bmod f$。

1. $u \leftarrow a, v \leftarrow f$。
2. $g_1 \leftarrow 1, g_2 \leftarrow 0$。
3. 当$u \neq 1$且$v \neq 1$时，重复执行
 3.1 当 z 整除 u 时，重复执行
 $u \leftarrow u/z$。
 若 z 整除 g_1，则 $g_1 \leftarrow g_1/z$；否则，$g_1 \leftarrow (g_1+f)/z$。
 3.2 当 z 整除 v 时，重复执行
 $v \leftarrow v/z$。
 若 z 整除 g_2，则 $g_2 \leftarrow g_2/z$；否则，$g_2 \leftarrow (g_2+f)/z$。
 3.3 若 $\deg(u) > \deg(v)$，则 $u \leftarrow u+v, g_1 \leftarrow g_1+g_2$；
 否则，$v \leftarrow v+u, g_2 \leftarrow g_2+g_1$。
4. 若 $u=1$，则返回(g_1)；否则，返回(g_2)。

步骤 3.3 中涉及多项式次数计算的表达式，可以简单地替换成多项式的二进制表达式的比较运算。这不同于算法 2.48，算法 2.48 中的步骤 3.1 直接进行次数计算。

殆逆算法

殆逆算法（算法 2.50）是二进制算法（算法 2.49）的一种改进，其中首先计算一个多项式 g 和一个正整数 k，使得

$$ag \equiv z^k \pmod{f}。$$

整理得到

$$a^{-1} = z^{-k} g \bmod f。$$

对于某些不被显式计算的 h_1 和 h_2，使得

$$ag_1 + fh_1 = z^k u$$
$$ag_2 + fh_2 = z^k v。$$

算法 2.50　求$\mathbb{F}_{2^m}$上元素殆逆的算法

输入：次数最多为 $m-1$ 的非零二进制多项式 a。

输出：$a^{-1} \bmod f$。

1. $u \leftarrow a, v \leftarrow f$。
2. $g_1 \leftarrow 1, g_2 \leftarrow 0, k \leftarrow 0$。
3. 当$u \neq 1$且$v \neq 1$时，重复执行
 3.1 当 z 整除 u 时，重复执行
 $u \leftarrow u/z, g_2 \leftarrow z \cdot g_2, k \leftarrow k+1$。
 3.2 当 z 整除 v 时，重复执行
 $v \leftarrow v/z, g_1 \leftarrow z \cdot g_1, k \leftarrow k+1$。
 3.3 若 $\deg(u) > \deg(v)$，则$u \leftarrow u+v, g_1 \leftarrow g_1+g_2$；
 否则，$v \leftarrow v+u, g_2 \leftarrow g_2+g_1$。
4. 若$u=1$，则$g \leftarrow g_1$；否则，$g \leftarrow g_2$。
5. 返回$(z^{-k}g \bmod f)$。

步骤5中的约减可以这样完成。令$l=\min\{i \geqslant 1 | f_i = 1\}$，其中$f(z)=f_m z^m+\cdots+f_1 z+f_0$。令 S 为多项式 g 的最右边 l 位形成的多项式，用 z^l 去除 $Sf+g$，得到$T=(Sf+g)/z^l$，T 的次数低于 m，因此$T=gz^{-l} \bmod f$。这一过程可重复直到获得$gz^{-k} \bmod f$为止。若 l 是某种极端值（取决于具体实现，如 $l \geqslant W$，W 是字长），则称约减多项式 f 是合适的，因为这样可使得约减的过程非常省事。

算法 2.50 中的步骤 3.1~3.2 比算法 2.49 中的简单。此外，算法中 g_1 和 g_2 在殆逆中增长得很慢。因此，如果约减多项式是合适的，那么可以期望算法 2.50 比算法 2.49 更优。反之亦然。与二进制算法一样，传统的次数计算可以换成简单的比较。

除法

二进制逆算法（算法 2.49）可以很容易地改进，以实现除法 $b/a=ba^{-1}$。在求逆与乘法的成本之比 I/M 较小的情况下，这对于椭圆曲线密码体制特别重要，因为对于应用仿射坐标的椭圆曲线点的运算（见 3.1.2 节）来说，用除法比用求逆和乘法更有效。

基于二进制算法的除法　为了得到b/a，算法 2.49 的步骤 2 被修改，用 $g_1 \leftarrow b$代替了 $g_1 \leftarrow 1$。相应的等式是

$$ag_1+fh_1=ub$$
$$ag_2+fh_2=vb。$$

当 $u=1$ 时算法终止，并得到 $g_1 = ba^{-1}$。我们期望这种除法算法能和二进制算法有相同的计算时间，因为 g_1 在算法 2.49 中只经过步骤 3.1 的少量迭代便达到了全长（g_1 的不同初始化对除法和求逆的时间作用不大）。

若二进制算法是求逆的应选方法，则仿射坐标的点运算采用除法更有效，因为这样倍点（或点加）的成本从 $I+2M$ 减少到了 $I+M$（I 和 M 分别表示实现求逆和乘法所用的时间）。若 I/M 较小，则这就成为一种重要的改进。例如，若 I/M 为3，则算法2.49 的一种除法算法变体将使仿射坐标的倍点或点加运算的时间减少20%。然而，若 $I/M>7$，则时间的节省将少于 12%。除非 I/M 非常小，否则采用减少求逆数量的方案是可能的，从而使得点乘只需要少量求逆，这便冲淡了应用除法算法的节省。

基于扩展 Euclidean 算法的除法 算法 2.48 可以转换成一个除法算法。然而初始化的变化，可能会对除法算法变体的实现有很大影响。注意有两个影响性能的问题，一个是变量长度的跟踪，另一个是在步骤 3.4 的加 g_1 的实现。

在算法 2.48 中有效地跟踪 u 和 v 的长度是非常容易的（长度短缩），而且还可以追踪 g_1 和 g_2 的长度。然而除法初始化的变化意味着 g_1 立即达到全长，并且呈现出缩短长度的优化。

第二个问题是在步骤 3.4 的加 g_1 的实现。一种实现方法是假设不约减的普通多项式加法能够实现，即假设 g_1 和 g_2 的次数都不超过 $m-1$。因为在初始化时 g_1 是全长的，所以步骤 3.4 的实现不太有效。

基于殆逆算法的除法 算法 2.50 与二进制算法类似，能够有效追踪 g_1 和 g_2 的长度（除了 u 和 v 的长度）可能是算法 2.50 与算法 2.49（提供合适的约减多项式 f）相比的优点。正如算法 2.48 的情况一样，这一优点在除法算法变体中消失了。

应当注意，在算法 2.50 中有效跟踪 g_1 和 g_2 的长度（除了 u 和 v 的长度）可能会引起严重的代码扩充（大概有 t^2 个碎片，而不是二进制算法中的 t 个碎片）。若不能容忍代码扩充（因为应用的限制或平台的限制），则殆逆可能比其他求逆算法更优越（即使约减多项式是适合的）。

2.4 最佳扩域算术

前面几节讨论了域 $\mathbb{F}_{p^m}$ 的算术运算，具体地讨论了 $p=2$ 的二进制域和 $m=1$ 的素域的算术运算。如 2.1 节注释的那样，二进制域的多项式基表示方法可以推广到所有扩域 $\mathbb{F}_{p^m}$ 上，其系数的运算在域 $\mathbb{F}_p$ 上进行。

从方便硬件实现来看，二进制域是最有吸引力的，因为其运算只涉及移位和按位的模 2 加。从实现的方便性来看，在通用处理器上用软件方法实现，是十分诱人的。然而这种软件实现的域乘法本质上一次只能计算少数几位，而且若可以利用硬件整数乘法器的话，则软件实现域的乘法可能比素域乘法要慢得多。另一方面，因为进位传播的限制，素域的算术运算很难有效实现。

最佳扩域的基本思想是选择 p 和 m，以及约减多项式，使之尽量适合硬件的特性。实际上，p 的选择应与单字相适应，以使进位容易处理（因为系数是单字）。

定义 2.51　最佳扩域（OEF）是满足下列条件的有限域 $\mathbb{F}_{p^m}$：

1. 于某些整数 n 和 c，$\text{lb}\,|c| \leqslant n/2$，有 $p = 2^n - c$。
2. $\mathbb{F}_p[z]$上存在一个既约多项式 $f(z) = z^m - \omega$。

若 $c \in \{\pm 1\}$，则 OEF 称为 1 型（若 $c = 1$，则 p 是梅森素数）。若 $\omega = 2$，则 OEF 称为 2 型。

1 型 OEF 在 $\mathbb{F}_p$ 的子域上具有特别简单的素数运算，而 2 型 OEF 可以简化扩域 $\mathbb{F}_{p^m}$ 的算术运算。表 2.1 给出了一些 OEF 的实例。

表 2.1　OEF 的参数示例。其中 $p = 2^n - c$ 是素数，$f(z) = z^m - \omega \in \mathbb{F}_p[z]$ 是 $\mathbb{F}_p$ 上的一个既约多项式。域 $\mathbb{F}_{p^m} = \mathbb{F}_p[z]/(f)$ 的阶近似为 2^{mn}

p	f	参数	类型
2^7+3	$z^{13}-5$	$n=7,\ c=-3,\ m=13,\ \omega=5$	—
$2^{13}-1$	$z^{13}-2$	$n=13,\ c=1,\ m=13,\ \omega=2$	I, II
$2^{31}-19$	z^6-2	$n=31,\ c=19,\ m=6,\ \omega=2$	II
$2^{31}-1$	z^6-7	$n=31,\ c=1,\ m=6,\ \omega=7$	I
$2^{32}-5$	z^5-2	$n=32,\ c=5,\ m=5,\ \omega=2$	II
$2^{57}-13$	z^3-2	$n=57,\ c=13,\ m=3,\ \omega=2$	II
$2^{61}-1$	z^3-37	$n=61,\ c=1,\ m=3,\ \omega=37$	I
$2^{89}-1$	z^2-3	$n=89,\ c=1,\ m=2,\ \omega=3$	I

下列结论可以用来确定给定的多项式 $f(z) = z^m - \omega$ 是否是 $\mathbb{F}_p[z]$ 上的一个既约多项式。

定理 2.52　令 m 是一个整数且 $m \geqslant 2$，$\omega \in \mathbb{F}_p^*$，则二项式 $f(z) = z^m - \omega$ 在 $\mathbb{F}_p[z]$ 上是既约多项式，当且仅当下面两个条件成立：

(i) m 的每一个素因子都能整除 ω 在 $\mathbb{F}_p^*$ 上的阶 e，但不能整除 $(p-1)/e$。

(ii) 若 $m \equiv 0 \pmod 4$，则 $p \equiv 1 \pmod 4$。

若ω的阶是$p-1$，则称ω为本原元。若ω为本原元且$m|(p-1)$，则容易验证满足定理 2.52 的两个条件。

推论 2.53 若ω是$\mathbb{F}_p^*$的本原元且$m|(p-1)$，则$z^m-\omega$是$\mathbb{F}_p[z]$上的既约多项式。

域$\mathbb{F}_{p^m}$的元素是多项式

$$a(z)=a_{m-1}z^{m-1}+\cdots+a_2z^2+a_1z+a_0,$$

其中系数a_i是$\mathbb{F}_p$上的元素。下面我们将给出一些实现 OEF 算术运算的算法。域运算的定时问题将在 5.1.5 节中给出。

2.4.1 加法和减法

若$a(z)=\sum_{i=0}^{m-1}a_iz^i$和$b(z)=\sum_{i=0}^{m-1}b_iz^i$是域$\mathbb{F}_{p^m}$的元素，则

$$a(z)+b(z)=\sum_{i=0}^{m-1}c_iz^i,$$

其中$c_i=(a_i+b_i)\bmod p$，即若$a_i+b_i\geqslant p$，则减p。类似可知域$\mathbb{F}_{p^m}$的减法。

2.4.2 乘法和约减

设$a,b\in\mathbb{F}_{p^m}$，a和b相乘可以先按$\mathbb{Z}[z]$上的多项式相乘（整数系数多项式相乘），然后系数在域$\mathbb{F}_p$上约减，并约减多项式f。这一乘法可形式地描述如下：

$$\begin{aligned}c(z)=a(z)b(z)&=\left(\sum_{i=0}^{m-1}a_iz^i\right)\left(\sum_{j=0}^{m-1}b_jz^j\right)\\&\equiv\sum_{k=0}^{2m-2}c_kz^k\equiv c_{m-1}z^{m-1}+\sum_{k=0}^{m-2}(c_k+\omega c_{k+m})z^k\pmod{f(z)},\end{aligned}$$

其中

$$c_k=\sum_{i+j=k}a_ib_j \bmod p。$$

Karatsuba-Ofman 技术可以用来约减域$\mathbb{F}_p$上的乘法。例如，

$$a(z)b(z) = (A_1 z^l + A_0)(B_1 z^l + B_0)$$
$$= A_1 B_1 z^{2l} + [(A_1 + A_0)(B_1 + B_0) - A_1 B_1 - A_0 B_0] z^l + A_0 B_0,$$

其中 $l = \lceil m/2 \rceil$，系数 A_0，A_1，B_0，B_1 是 $\mathbb{F}_p[z]$ 上的次数低于 l 的多项式。这一处理过程可以重复，虽然对于小的 m 值考虑二叉之外的分裂法可能更有利。素域的类似情况曾在 2.2.2 节中讨论过。

域 $\mathbb{F}_p$ 上的约减

域 $\mathbb{F}_p$ 上的约减的最直接实现是在计算每一个 c_k 时直接完成加法和乘法的计算。$\mathrm{lb}_2 |c| \leqslant n/2$ 的限制意味着域 $\mathbb{F}_p$ 上的约减只需要少量的简单运算。算法 2.54 利用移位、加法和单精度乘法实现了基数 B 的约减运算。

算法 2.54　模 $M = B^n - c$ 的约减

输入：基数 B，正整数 x，模 $M = B^n - c$，其中 c 是一个 l 位的基为 B 的正整数，对于某些 $l < n$。

输出：$x \bmod M$。

1. $q_0 \leftarrow \lfloor x/B^n \rfloor, r_0 \leftarrow x - q_0 B^n$。　　$\{x = q_0 B^n + r_0$，其中 $r_0 < B^n\}$
2. $r \leftarrow r_0, i \leftarrow 0$。
3. 当 $q_i > 0$ 时，重复执行
 3.1　$q_{i+1} \leftarrow \lfloor q_i c / B^n \rfloor$。　　$\{q_i c = q_{i+1} B^n + r_{i+1}$，其中 $r_{i+1} < B^n\}$
 3.2　$r_{i+1} \leftarrow q_i c - q_{i+1} B^n$。
 3.3　$i \leftarrow i + 1, r \leftarrow r + r_i$。
4. 当 $r \geqslant M$ 时，$r \leftarrow r - M$。
5. 返回(r)。

注 2.55　（算法 2.54 的执行细节）

(i) 若 $l \leqslant n/2$ 且 x 最多有 $2n$ 位的基为 B 的数，则算法 2.54 的步骤 3.1 最多执行两次（即最多两次用 c 乘）。

(ii) 作为一种选择，在每一阶段都可用商和余数来表示 x。步骤 1~4 如下重复：

1. 当 $x \geqslant B^n$ 时，重复执行
 1.1　记 $x = v B^n + u$，其中 $u < B^n$。
 1.2　$x \leftarrow cv + u$。

2. 若 $x \geqslant M$，则 $x \leftarrow x - M$。

(iii) 算法 2.54 可以修改，以处理 $M = B^n + c$ 的情况，其中 c 是小于 B^{n-1} 的某些正整数。在步骤 3.3 中用 $r \leftarrow r + (-1)^i r_i$ 代替 $r \leftarrow r + r_i$，并修改步骤 4，也能处理 $r < 0$ 的情况。

对于最佳扩域，$B = 2$ 时可使用算法 2.54，在 $x < 2^{2n}$ 的情况下算法 2.54 最多需要两次用 c 乘。当 $c = 1$（1 型 OEF）且 $x \leqslant (p-1)^2$ 时，约减如下进行：

记 $x = 2^n v + u$；$x \leftarrow v + u$；若 $x \geqslant p$，则 $x \leftarrow x - p$。

1 型 OEF 特别具有吸引力，因为 $\mathbb{F}_p$ 上的乘法（带约减）能够用一个乘和少量其他运算来完成。然而，约减 p 的运算可能成为域 $\mathbb{F}_{p^m}$ 上的乘法的最耗时部分，而直接使用乘-累加的策略减少约减的数量可能是有效的。

累加和约减

通过采用对 $c(z)$ 的系数进行累加的策略，可以减少计算域 $\mathbb{F}_{p^m}$ 上的乘积 $c(z) = a(z)b(z)$ 时所需的域 $\mathbb{F}_p$ 上的约减数量。因为 $f(z) = z^m - \omega$，所以乘积可以写为

$$c(z) = a(z)b(z) \equiv \sum_{k=0}^{2m-2} c_k z^k \equiv \sum_{k=0}^{m-1} c_k z^k + \omega \sum_{k=m}^{2m-2} c_k z^{k-m}$$

$$\equiv \sum_{k=0}^{m-1} \Bigg(\underbrace{\sum_{i=0}^{k} a_i b_{k-i} + \omega \sum_{i=k+1}^{m-1} a_i b_{k+m-i}}_{c'_k} \Bigg) z^k \pmod{f(z)}。$$

若把系数 c'_k 表示成 $\mathbb{Z}$ 中的整数（作一个为整数，不对模 p 约减），则只要执行一次约减（而不是 m 次约减）就可完成 $c'_k \mod p$ 的计算。这样做带来的一个麻烦是，需要进行累加 c'_k 的多字的运算（加法和用 ω 乘）。

与对每一操作直接约减的方法相比，应当注意，对域 $\mathbb{F}_p$ 上每一运算都完全约减是不必要的。例如，只把最终结果约减为一个单字的值就足够了。然而，对一个单字或小于 2^n 的一个值频繁地约减是很浪费时间的，特别是需要处理进位或比较运算时就更显费时。

与 p 的值相关，乘-累加方法要应用两个或三个累加寄存器（假设 p 存储在一个寄存器中）。与实现素域的运算类似，域 $\mathbb{F}_{p^m}$ 上的乘法运算的成本期望能与素域 $\mathbb{F}_q$ 上乘法运算的成本相当，其中 $q \approx p^m$ 且其可以快速约减（如 NIST 推荐的素数，见 2.2.6 节）。

对于 $c'_k \mod p$ 的约减，注意

$$c'_k \leqslant (p-1)^2 + \omega(m-1)(p-1)^2 = (p-1)^2(1+\omega(m-1))。$$

若 $p = 2^n - c$ 使得

$$\text{lb}(1+\omega(m-1)) + 2\ \text{lb}|c| \leqslant n, \tag{2.4}$$

则约减最多需要两次用 c 乘就可完成。作为一个例子，若 $p = 2^{28} - 165$且$f(z) = z^6 - 2$，则

$$\text{lb}(1+\omega(m-1)) + 2\ \text{lb}\,|c| = \text{lb}\ 11 + 2\ \text{lb}165 < n = 28$$

且条件(2.4)满足。

若累加是在多个字宽为 W 位的寄存器中进行，则选择 $p = 2^n - c$且 $n < W$可使若干项的累加计算只在两个寄存器中进行（不需要第三个寄存器，也不需要局部约减）。例如，对于 $W = 32$，$p = 2^{28} - 165$ 是有吸引力的选择。然而，把这种方法与整数乘的最佳应用相比，若需要很大的 m 以获得足够大的域，则这种方法不一定是有效的。

例 2.56 （*累加方法*）考虑由 $p = 2^{31} - 1$和 $f(z) = z^6 - 7$ 所定义的最佳扩域 OEF 且机器字长 W=32 的情况。因为这是 1 型 OEF，子域的约减特别简单，因此局部约减和累加结合在计算$c'_k \bmod p$ 时是十分有效的。虽然在每次运算后都进行约减使之能进入单个寄存器可能是麻烦费时的，但是应用两个寄存器或三个寄存器进行累加却是可以的。

设累加器由两个寄存器组成。通过把 $a_i b_j$ 写成 $a_i b_j = 2^{32}v + u$并且把 $2v + u$ 加到累加器上，对组成 $a_i b_j$ 的每一项进行部分约减是可以实现的。类似地，在加了乘积 $a_i b_j$ 之后，累加器自身可以被局部约减。

若累加器是 3 个字长的，则局部约减就不必要了，并且累加器的一部分仅涉及两个寄存器。另一方面，用 $\omega = 7$ 乘并最后约减是相当复杂的（与两个寄存器方案相比）。

乘-累加的方法还可用于域$\mathbb{F}_{p^m}$上的开方。开方总共需要 $m + \binom{m}{2} = m(m+1)/2$ 次整数乘（和可能的 $m-1$ 次用 ω乘）。域$\mathbb{F}_p$上的约减运算的成本取决于所用的方法，特别地，若只用单次约减来寻找c'_k，则约减的次数就与普通乘法的次数相同。

2.4.3　求逆

设$a \in \mathbb{F}_{p^m}$，$a \neq 0$，求 a 的逆就是寻找一个多项式$a^{-1} \in \mathbb{F}_{p^m}$，使得 $aa^{-1} \equiv 1 \pmod f$。对于 OEF 可以利用改进的 Euclidean 算法求逆。然而，Frobenius 映射和一种特殊形式的 f 可以用来获得最快的求逆方法。一旦域的乘法已经实现，则这种方法简单高效，这是因为把域$\mathbb{F}_{p^m}$上的求逆简化成子域$\mathbb{F}_p$上的求逆只需要很少量的乘法。

算法 2.59 计算

$$a^{-1} = (a^r)^{-1} a^{r-1} \bmod f \ , \tag{2.5}$$

其中，

$$r = \frac{p^m - 1}{p-1} = p^{m-1} + \cdots + p^2 + p + 1。$$

因为$(a^r)^{p-1} \equiv 1 \pmod{p^m}$，这使得$a^r \in \mathbb{F}_p$。因此，一种合适的算法可以用于域$\mathbb{F}_p$上的求逆，以便计算式(2.5)中的$(a^r)^{-1}$。

式(2.5)中的$a^{r-1} = a^{p^{m-1}+\cdots+p}$可以通过以下定义的 Frobenius 映射$\varphi$的性质来实现：

$$\varphi : \mathbb{F}_{p^m} \to \mathbb{F}_{p^m}, \ \varphi(a) = a^p。$$

根据这一映射，域$\mathbb{F}_p$的元素是固定的。因此，若$a = a_{m-1}z^{m-1} + \cdots + a_2 z^2 + a_1 z + a_0$，则

$$\varphi^i : a \mapsto a_{m-1}z^{(m-1)p^i} + \cdots + a_1 z^{p^i} + a_0 \bmod f。$$

为了计算z的幂 modf，把非负整数e写为$e = qm + r$，其中$q = \lfloor e/m \rfloor$，$r = e \bmod m$。因为$f(z) = z^m - \omega$，这使得

$$z^e = z^{qm+r} \equiv \omega^q z^r \pmod{f(z)}。$$

注意，若$p \equiv 1 \pmod{m}$，则求$\varphi^i(a)$的值的计算就很简单。根据定理 2.52，m的每一个素因子都整除$p-1$。必要地，若m是无平方因子数，则条件$p \equiv 1 \pmod{m}$成立。这些结论总结于定理 2.57。

定理 2.57 （Frobenius 映射迭代的作用）给定一个 OEF，设$p = 2^n - c$，$f(z) = z^m - \omega$，令域$\mathbb{F}_{p^m}$上的 Frobenius 映射定义为$\varphi : a \mapsto a^p \bmod f$，则

(i) φ的第i次迭代是映射

$$\varphi^i : a \mapsto \sum_{j=0}^{m-1} a_j \omega^{\lfloor jp^i/m \rfloor} z^{jp^i \bmod m}。$$

(ii) 若m是无平方因子数，则$p \equiv 1 \pmod{m}$，因此$jp^i \bmod m = j$对所有$0 \leqslant j \leqslant m-1$都成立。

可以对于感兴趣的$e = jp^i$预先计算出$z^e \equiv \omega^{\lfloor e/m \rfloor} z^{e \bmod m} \pmod{f(z)}$的值，此种情况下求$\varphi^i(a)$的值的计算可以用$m-1$次域$\mathbb{F}_p$上的乘法来完成。应用加法链和少量的域乘法以及$\varphi^i$，可以有效地找出式(2.5)中的$a^{r-1}$。

例 2.58　（计算 a^{r-1}）设有一个由参数 $p=2^{31}-1$ 和 $f(z)=z^6-7$ 所定义的 OEF。于是有 $r-1=p^5+p^4+\cdots+p$。我们按表 2.2 所示的顺序（加法链类的方法）计算 a^{r-1}，其中 $m=6$。φ 和 φ^2 的值使用表 2.3 中根据定理 2.57 预计算的值。

表 2.2　对于 $r=\frac{p^m-1}{p-1}$，$m\in\{3,5,6\}$ 计算 a^{r-1}。最后一行给出了用 $\mathbb{F}_{p^m}$ 上的乘法（M）和 Frobenius 映射（φ）迭代综合表示的计算开销

$m=3$	$m=5$	$m=6$
$T\leftarrow a^p$ $T\leftarrow Ta=a^{p+1}$ $a^{r-1}\leftarrow T^p=a^{p^2+p}$	$T_1\leftarrow a^p$ $T_1\leftarrow T_1a=a^{p+1}$ $T_2\leftarrow T_1^{p^2}=a^{p^3+p^2}$ $T_1\leftarrow T_1T_2=a^{p^3+p^2+p+1}$ $a^{r-1}\leftarrow T_1^p$	$T_1\leftarrow a^p$ $T_2\leftarrow T_1a=a^{p+1}$ $T_3\leftarrow T_2^{p^2}=a^{p^3+p^2}$ $T_2\leftarrow T_3T_2=a^{p^3+p^2+p+1}$ $T_2\leftarrow T_2^{p^2}=a^{p^5+p^4+p^3+p^2}$ $a^{r-1}\leftarrow T_2T_1$
开销：$1M+2\varphi$	开销：$2M+3\varphi$	开销：$3M+3\varphi$

表 2.3　在 $p=2^{31}-1$，$f(z)=z^6-7$ 且 $i\in\{1,2\}$ 的情况下，φ^i 值的预计算（见例 2.58）。若 $a=a_5z^5+\cdots+a_1z+a_0\in\mathbb{F}_{p^6}$，则 $\varphi^i(a)=a^{p^i}\equiv\sum_{j=0}^{5}a_j\omega^{\lfloor jp^i/m\rfloor}z^j \pmod f$

$z^{jp}\equiv\omega^{\lfloor jp/m\rfloor}z^j \pmod f$	$z^{jp^2}\equiv\omega^{\lfloor jp^2/m\rfloor}z^j \pmod f$
$z^p\equiv 1\ 513\ 477\ 736z$	$z^{p^2}\equiv 1\ 513\ 477\ 735z$
$z^{2p}\equiv 1\ 513\ 477\ 735z^2$	$z^{2p^2}\equiv 634\ 005\ 911z^2$
$z^{3p}\equiv 2\ 147\ 483\ 646z^3\equiv -1z^3$	$z^{3p^2}\equiv 1z^3$
$z^{4p}\equiv 634\ 005\ 911z^4$	$z^{4p^2}\equiv 1\ 513\ 477\ 735z^4$
$z^{5p}\equiv 634\ 005\ 912z^5$	$z^{5p^2}\equiv 634\ 005\ 911z^5$

一般来说，若 $w(x)$ 是整数 x 的汉明重量，则 a^{r-1} 可以用

$$t_1(m)=\lfloor \mathrm{lb}\,(m-1)\rfloor+w(m-1)-1$$

次 $\mathbb{F}_{p^m}$ 上的乘法和

$$t_2(m)=\begin{cases}t_1(m)+1, & m\text{ 为奇数}\\ j=\lfloor \mathrm{lb}\,(m-1)\rfloor+1, & \text{对于某些 } j,\ m=2^j\\ \lfloor \mathrm{lb}(m-1)\rfloor+w(m)-1, & \text{其他}\end{cases}$$

次的 Frobenius 映射（φ）迭代而计算出来。因为 $t_2(m)\leqslant t_1(m)+1$，因此对于 $m>2$ 计算 a^{r-1} 的时间取决于 $\mathbb{F}_{p^m}$ 上的乘法（$\mathbb{F}_{p^m}$ 上的一次乘法比计算 φ^i 所需要的 $\mathbb{F}_p$ 上的 $m-1$ 次乘法都要费时）。

算法 2.59 OEF 求逆

输入：$a \in \mathbb{F}_{p^m}$，$a \neq 0$。

输出：元素 $a^{-1} \in \mathbb{F}_{p^m}$，满足 $aa^{-1} \equiv 1 \pmod{f}$。

1. 利用加法链方法寻找 a^{r-1}，其中 $r = (p^m - 1)/(p-1)$。
2. $c \leftarrow a^r = a^{r-1}a \in \mathbb{F}_p$。
3. 利用 $\mathbb{F}_p$ 上的一个求逆算法获得 c^{-1}，使得 $cc^{-1} \equiv 1 \pmod{p}$。
4. 返回 $(c^{-1}a^{r-1})$。

注 2.60 （算法 2.59 的执行细节）

(i) 算法 2.59 的步骤 2 中的元素 $c \in \mathbb{F}_p$。因此只需要完成常数项 $a^{r-1}a$ 的算术运算（需要 m 次 $\mathbb{F}_p$ 上的乘法和一次用 ω 乘）。

(ii) 因为 $c^{-1} \in \mathbb{F}_p$，所以在步骤 4 中的乘法需要 m 次 $\mathbb{F}_p$ 上的乘运算。

(ii) 执行时间取决于寻找 a^{r-1} 所需的 $t_1(m)$ 次 $\mathbb{F}_{p^m}$ 上的乘法和步骤 3 求子域逆的开销。

求逆的开销与乘法的成本之比 *I/M* 是基本的着眼点。当 $m = 6$ 时算法 2.59 所花费的时间远大于在寻找 a^{r-1} 时的 $t_1(6) = 3$ 次乘法，因为步骤 3 的子域求逆的时间是主要的。然而对于通用处理器，这一比率期望远小于素域 $\mathbb{F}_q$ (其中 $q \approx p^m$)上的比率。

2.5 注释和进一步参考

2.1 节

要了解有限域理论的基础，请参看 Koblitz 的[254]和 McEliece 的[311]。Lidl 和 Niederreiter 的[292]给出了更全面的研究。

2.2 节

Menezes, van Oorshot 和 Vanstone 的[319]简明地介绍了密码学中感兴趣的二进制算术和模整数算术的算法。Knuth 的[249]是本标准参考书。Koç 的[258]论述了一些模乘方法，包括经典的方法和 Karatsuba-Ofman 方法（一种乘法与约减交错的方法）以及 Montgomery 乘法。

选择运算数扫描方式的基本乘法（算法 2.9）还是选择积扫描方式的乘法（算法 2.10）取决于所用的平台。一般而言，算法 2.9 访问存储器较多，而算法 2.10 的程序控制较复杂，除非把循环打开。Comba[101]对 16 位的 Intel 80286 处理器就此进行了详细的比较，打开循

环的积扫描方式的乘法程序比 OpenSSL 的“comba”程序更灵巧。

Scott[416]讨论了在三种 32 位 Intel IA-32 处理器（80486，Pentium 和 Pentium Pro）环境下的乘法运算，并且给出了用运算数扫描、积扫描（Comba 方法）、基于积扫描的 Karatsuba-Ofman 方法和浮点硬件等方法进行模幂计算的实验结果。基于较新的 IA-32 处理器的乘法实验在 5.1.3 节讨论。Dussé 和 Kaliski 指出[127]，在 Motorola 数字信号处理器 56000 环境下，算法 2.9 的内循环中变量 U 的抽取比较费时。处理器有 56 位累加器，但只做 24 位带符号乘，而且在 Montgomery 乘法中采用积扫描技术，据报道是相当快的。

Karatsuba-Ofman 乘法是由 Karatsuba 和 Ofman 在[239]中提出的。对于比较小的整数，Karatsuba-Ofman 的变体方法对于乘法的节省是不够的，不足以使这种方法与最好的经典方法相比也有竞争力。Knuth[249]和 Koç [258]对 Karatsuba-Ofman 方法做了更详细的介绍。

Barrett 约减方法（算法 2.14）是由 Barrett 在[29]中提出的。Bosselaers, Govaerts 和 Vandewalle 对此给出了描述[66]，并给出了经典约减方法、Barrett 约减方法和 Montgomery 约减方法的比较结果。他们的结果表明，若不把预计算和数的转换的时间计算在内，则这些方法在开销上基本类似，Montgomery 的约减方法最快，而经典的约减方法比 Barrett 约减方法稍慢。运算量的比较由在便携式 80386 计算机上实际执行的结果所证实。据 De Win, Mister, Preneel 和 Wiener 在[111]中的报道，在 Intel Pentium Pro 计算机上进行 192 位素域 $\mathbb{F}_p$ 的域运算时，Montgomery 约减方法和 Barrett 约减方法的差别是可忽略的。

Montgomery 约减方法是 Montgomery 在[330]中提出的。Koç, Acar 和 Kaliski 分析了五个 Montgomery 乘法算法[260]。根据是采用运算数扫描进行乘法还是采用积扫描方式进行乘法，而将这一方法分类为单独约减方式和约减与乘法相结合的方式。在算法测试中，他们得出如下结论：一种俗称为“综合运算数扫描”的方法（通过一个运算数进行外循环，而且在外循环的每个下标都是约减运算跟着一个乘法运算）是最简单的，而且对于通用处理器来说是最好的。Koç 和 Acar 把 Montgomery 乘法推广到了二进制域[259]。

二进制最大公因子算法（算法 2.21）是由 Stein 在[451]中提出的，并且 Knuth 在[249]中对此进行了分析。Bach 和 Shallit 在[23]中全面分析了几种求解最大公因子的算法。二进制求逆算法（算法 2.22）是根据相应的扩展 gcd 算法改写而成的。

Lehmer 提出了一种经典 Euclidean 算法的变体[278]，这种变体方法用单精度运算代替了昂贵的多精度除法。Knuth 详细考察了这一算法[249]，Sorenson 分析了这一算法的一种细小的改进[450]。Durand[126]简单并综合介绍了由扩展 Euclidean 算法改进而来的各种求逆算法，包括二进制 gcd 算法、Lehmer 算法以及在 SGS-Thomson 公司的 32 位 RISC 处理器（智能卡用的处理器）环境下进行 RSA 和椭圆曲线点乘的时间分析。对于这种处理器，

Lehmer 算法表现出了很大的优越性，实际上用它进行点乘的速度快于采用投影坐标情况下的点乘速度。

部分 Montgomery 逆方法（算法 2.23）是由 Kaliski 在[234]中提出的。De Win, Mister, Preneel 和 Wiener 在[111]中指出，基于这种算法的一种求逆方法优于扩展 Euclidean 算法（算法 2.19）的许多变体。他用 Intel 的 Pentium Pro 处理器进行了实验测试，但是实验细节没有发表。Savas 和 Koç 给出了算法 2.25 的一种推广[403]，还给出了一种求普通逆的类似算法。

同时求逆的算法（算法 2.26）是由 Montgomery 提出的[331]，这种技术常被用于因子分解的椭圆曲线方法（ECM）。Cohen 在[99，算法 10.3.31]中给出了算法 2.26 的一种扩展，该扩展也被用于椭圆曲线方法（ECM）。

NIST 素数（见 2.2.6 节）是由联邦信息处理标准（FIPS）关于数字签名标准的文件 186-2 给出的[140]。在这一文件中推荐美国政府采用椭圆曲线密码。Solinas[445]讨论了梅森数 2^k-1 的推广。这种推广的梅森数可以加速约减运算（不用除法）；NIST 素数是它的特例。

2.3 节

关于多项式乘法的算法 2.35 和算法 2.36 是由 López 和 Dahab 在[301]中提出的。他们的工作扩展了 Lim 和 Lee 在[295]中提出的“comb”求幂方法。运算量的比较和执行（Intel 类处理器或 Sun UltraSPARC 处理器）的结果说明，在存储器要求适当的情况下算法 2.36 大大快于算法 2.34。在注 2.37 中的多表变体方法也是由 López 和 Dahab 在[301]中提出的。

在注 2.38 中提到的 Sun 公司微系统实验室开发的 OpenSSL 的作者是 Sheueling Chang Shantz 和 Douglas Stebila。我们的注释是基于 OpenSSL–0.9.8 的一种简介。Weimerskirch, Stebila 和 Chang Shantz 讨论了一种有限域算术运算的重要技巧[478]。附录 C 有少量关于 OpenSSL 库的注释。

NIST 约减多项式（见 2.3.5 节）是由联邦信息处理标准（FIPS）关于数字签名标准的文件 186-2 给出的[140]。在这一文件中推荐美国政府采用椭圆曲线密码。

二进制逆算法（算法 2.49）与多项式逆算法（算法 2.22）相对应。Almost 逆算法（算法 2.50）是由 Schroeppel, Orman, O'Malley 和 Spatscheck 在[415]中提出的。Kaliski 在[234]中提出了一种在 Montgomery 逆中用到的类似算法（算法 2.23）。

Goodman 和 Chandrakasan[177], Chang Shantz[90], Durand[126]和 Schroeppe[412]分别介绍了域除法的算法。求逆算法和除法算法对编译器和处理器的特性十分敏感，对运算量的粗略分析会产生误导。Fong, Hankerson, López 和 Menezes[144]讨论了求逆算法和除法算法，并且针对 Intel Pentium III 处理器和 Sun UltraSPARC 上所用的几种编译器进行了时间比较分析。

在正则基表示法中，有限域 $\mathbb{F}_{2^m}$ 上的元素用基 $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ 中的项来表示。正则基表示法的一个优点是，域元素的平方运算是对它的向量表示的简单旋转。Mullin, Onyszchuk, Vanstone 和 Wilson[337]介绍了最佳正则基的概念及其在降低域 $\mathbb{F}_{2^m}$ 上的元素乘法的硬件实现复杂性中的应用。Agnew, Mullin, Onyszchuk 和 Vanstone[6]以及 Sunar 和 Koç [456]分别介绍了利用最佳正则基实现域 $\mathbb{F}_{2^m}$ 的算术运算。

Ash, Blake 和 Vanstone[19]进一步研究了低复杂性的正则基，低复杂性的正则基也称为 Gaussian 正则基。Gaussian 正则基在 ANSI X9.62 关于 ECDSA 的标准[14]中有明确的描述。经验表明，对于硬件实现最佳正则基不不比多项式基有显著的优点。况且，与多项式基的乘法相比较，正则基的乘法的软件实现非常慢。关于这一点，请参看 Reyhani-Masoleh 和 Hasan 的[390]以及 Ning 和 Yin 的[348]。

2.4 节

Bailey 和 Paar 在[25, 26]中介绍了最佳扩域。定理 2.52 取自 Lidl 和 Niederreiter 的[292]中的定理 3.75。定理 2.57 纠正了[26]中的推论 2。[26]中的 OEF 构造算法在不可约性测试中有一个小缺陷，导致在它们的 2 型 OEF（如当 $p = 2^8 - 5$ 时，$z^{25} - 2$ 就不是既约的）表中产生了一个新的错误入口。由 Bailey 和 Paar 给出的 2.4.3 节中的求逆方法基于 Itoh 和 Tsujii 的[217]，还可参考[183]。

Lim 和 Hwang[293]对各种优化方法给出了全面的介绍，并且对 Intel 和 DEC 处理器给出了有用的基准时间分析。他们的运算量的分析有利于一种用于求逆的基于算法 2.59 的 Euclidean 算法的变体。然而，在这一层次上对运算进行粗略地计量经常不能充分反映处理器或编译器的特性，在后来的工作[294]中他们指出，算法 2.59 在 Intel Pentium II 和 DEC Alpha 处理器上的执行是相当快的。Chung, Sim 和 Lee 指出[97]，在[26]中给出的逆运算中所需的 Frobenius 映射的数的量没有必要是最小的。他们给出了一个修订公式以及求逆算法的比较，并且对于韩国三星公司的一种低档 CalmRISC 8 位处理器（带有数学协处理器）给出了执行结果。

第 3 章　椭圆曲线算术

基于椭圆曲线的密码体制取决于椭圆曲线的点的算术运算。如第 2 章的注释所指出的那样，椭圆曲线算术根据其所依赖的域的运算来定义，运算的效率是关键。因此，椭圆曲线算术运算的高效实现就是至关重要的问题。

图 3.1 给出了椭圆曲线数字签名算法（ECDSA，4.4.1 节）的协议模型框架。椭圆曲线的算术运算不仅仅依赖于域的算术运算，而且还依赖于大数的模运算（例如，若采用 Koblitz 曲线，则还涉及τ-adic 运算，见 3.4 节）。ECDSA 还应用杂凑函数和一些模运算，但是最费时的运算步骤还是椭圆曲线的算术运算。

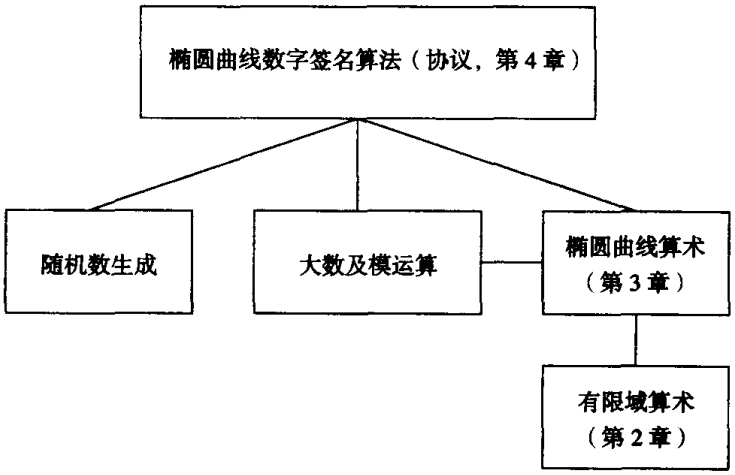


图 3.1　ECDSA 模型框架

3.1 节将介绍椭圆曲线的概念、椭圆曲线点群的加法和倍点运算、群的结构及其性质。3.2 节将给出椭圆曲线点的投影坐标表示法，以及相应的点加和倍点运算算法，特别注意了当域的求逆运算比域的乘法运算更费时的情况。3.3 节将讨论点乘的方法，点乘运算决定着椭圆曲线密码体制的实现效率。

3.4 节、3.5 节和 3.6 节所给出的方法在以下意义上是互相联系的，即它们都利用椭圆曲线的自同态来降低椭圆曲线点乘运算中倍点运算的开销。3.4 节讨论特殊的 Koblitz 曲线，

此时$\mathbb{F}_2$上的曲线倍点运算被替换成简单的域平方运算。3.5 节将研究更广泛的允许自同态的椭圆曲线类，它们可以有效地减少椭圆曲线点乘运算中倍点运算的数量。3.6 节将研究二进制域上的椭圆曲线，其运算方法采用快速的折半运算替换了大部分的倍点运算。3.7 节将对所用点乘方法的运算量进行比较。3.8 节将给出本章的结束语以及注释和参考读物。

3.1　椭圆曲线的概念

定义 3.1　域 K 上的椭圆曲线 E 由下述方程定义：

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{3.1}$$

其中$a_1, a_2, a_3, a_4, a_6 \in K$且$\Delta \neq 0$，$\Delta$是 E 的判别式，具体定义如下：

$$\left.\begin{aligned} \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{aligned}\right\} \tag{3.2}$$

若 L 是 K 的扩域，则 E 上的 L 有理数点的集合是

$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\}$，其中 ∞ 是无穷远点。

备注 3.2　（定义 3.1 的注释）

(i) 式(3.1) 称为 Weierstrass 方程。

(ii) 我们称 E 是域 K 上的椭圆曲线，这是因为系数 a_1，a_2，a_3，a_4，a_6均为域 K 的元素。有时我们将椭圆曲线记为 E/K 以强调椭圆曲线 E 定义在域 K 上，并称 K 为 E 的基础域。注意，若椭圆曲线 E 定义在域 K 上，则 E 也定义在 K 的扩域上。

(iii) 条件$\Delta \neq 0$ 确保椭圆曲线是“光滑”的，即曲线的所有点都没有两个或两个以上不同的切线。

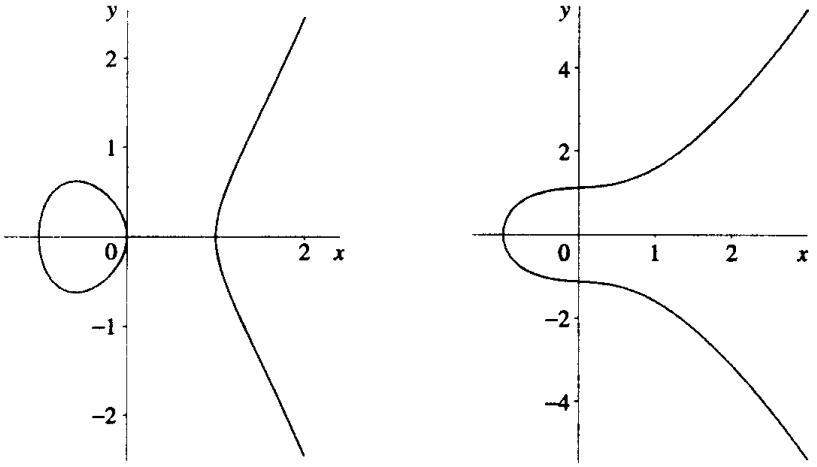
(iv) 点∞是曲线的惟一的一个无穷远点，它满足投影形式的 Weierstrass 方程（见 3.2 节）。

(v) 曲线 E 的 L 有理数点是满足曲线方程且坐标 x 和 y 属于 L 的点(x, y)，并认为无穷远点是 K 的所有扩域 L 上的 L 有理数点。

例 3.3 （ℝ上的椭圆曲线）考虑定义在实数域ℝ上的椭圆曲线

$$E_1 : y^2 = x^3 - x,$$
$$E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4},$$

其点 $E_1(\mathbb{R})\backslash\{\infty\}$和 $E_2(\mathbb{R})\backslash\{\infty\}$画在图 3.2 中。



(a) $E_1 : y^2 = x^3 - x$　　(b) $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

图 3.2　ℝ上的椭圆曲线

3.1.1 简化的 Weierstrass 方程

定义 3.4　由 Weierstrass 方程给出的定义在域 K 上的两个椭圆曲线 E_1和 E_2

$$E_1 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$E_2 : y^2 + \bar{a}_1xy + \bar{a}_3y = x^3 + \bar{a}_2x^2 + \bar{a}_4x + \bar{a}_6$$

被称为在域 K 上是同构的，若存在 $u, r, s, t \in K$且 $u \neq 0$，使得变量变换

$$(x, y) \to (u^2x + r, u^3y + u^2sx + t) \tag{3.3}$$

把方程E_1变成方程 E_2。式 (3.3) 的变换称为变量的相容性变换。

定义在域 K 上的一个 Weierstrass 方程

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

能够用变量的相容性变换来简化。这种简化的方程将应用于本书的其余部分。我们将分别考虑域 K 的特征等于 2 或 3 和域的特征不等于 2 或 3 两种情况。

1．若域 K 的特征不等于 2 或 3，则变量的相容性变换

$$(x,y)\to\left(\frac{x-3a_1^2-12a_2}{36},\frac{y-3a_1x}{216}-\frac{a_1^3+4a_1a_2-12a_3}{24}\right)$$

把 E 变换为曲线

$$y^2=x^3+ax+b, \tag{3.4}$$

其中 $a,b\in K$。曲线的判别式是 $\Delta=-16(4a^3+27b^2)$。

2. 若域 K 的特征是 2，则有两种情况要考虑。若 $a_1\neq 0$，则变量的相容性变换

$$(x,y)\to\left(a_1^2x+\frac{a_3}{a_1},a_1^3y+\frac{a_1^2a_4+a_3^2}{a_1^3}\right)$$

把 E 变换为曲线

$$y^2+xy=x^3+ax^2+b, \tag{3.5}$$

其中 $a,b\in K$。这样的曲线称为非超奇异的（见定义 3.10），且判别式为 $\Delta=b$。若 $a_1=0$，则变量的相容性变换

$$(x,y)\to(x+a_2,y)$$

把 E 变换为曲线

$$y^2+cy=x^3+ax+b, \tag{3.6}$$

其中 $a,b,c\in K$。这样的曲线称为超奇异的（见定义 3.10），且判别式为 $\Delta=c^4$。

3. 若域 K 的特征是 3，则也有两种情况要考虑。若 $a_1^2\neq -a_2$，则变量的相容性变换

$$(x,y)\to\left(x+\frac{d_4}{d_2},y+a_1x+a_1\frac{d_4}{d_2}+a_3\right),$$

其中 $d_2=a_1^2+a_2$，$d_4=a_4-a_1a_3$，把 E 变换为曲线

$$y^2=x^3+ax^2+b, \tag{3.7}$$

其中 $a,b\in K$。这样的曲线称为非超奇异的，且判别式为 $\Delta=-a^3b$。若 $a_1^2=-a_2$，则变量的相容性变换

$$(x,y)\to(x,y+a_1x+a_3)$$

把 E 变换为曲线

$$y^2=x^3+ax+b, \tag{3.8}$$

其中 $a,b\in K$。这样的曲线称为超奇异的（见定义 3.10），并且判别式为 $\Delta=-a^3$。

3.1.2 群的运算法则

令 E 是一个定义在域 K 上的椭圆曲线。根据“弦和切线”法则，$E(K)$上的两个点相加得到 $E(K)$上的第三个点。点集合 $E(K)$及其这种加法运算构成一个加法交换群，并且∞为其无穷远点。就是这个群被用来构建椭圆曲线密码体制。

群的加法规则最好用几何方法说明。令 $P=(x_1, y_1)$ 和 $Q=(x_2, y_2)$ 是椭圆曲线 E 上的两个不同的点，则 P 与 Q 的和 R 如下定义。首先画一条连接 P 和 Q 的直线，这条直线与椭圆曲线相交于第三点，则这个交点关于 x 轴的对称点就是 R 点。这一几何表示示于图 3.3(a) 中。

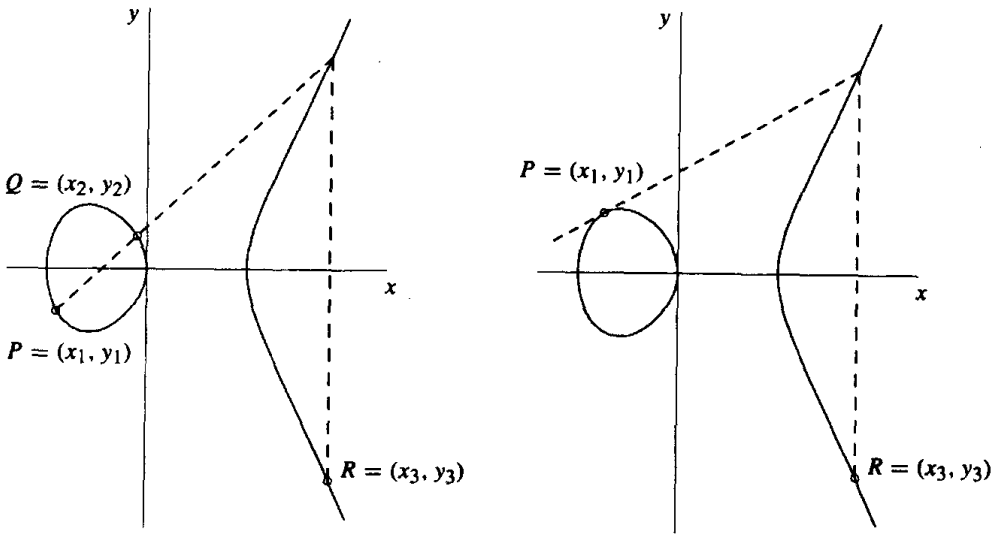
按如下方法求出 P 的倍点 R。首先在 P 点画椭圆曲线的切线，这条切线与椭圆曲线相交于第二点，这个交点关于 x 轴的对称点就是 R 点。这一几何表示示于图 3.3(b)中。

可以从上述几何描述得出群运算的代数公式。下面，我们分别对于以下三种情况给出对应于式(3.4)的简化 Weierstrass 方程的群运算代数公式：

(i) 基础域 K 的特征不等于 2 或 3（如$K=\mathbb{F}_p$，p 是大于 3 的素数）。

(ii) 式 (3.5) 的非超奇异椭圆曲线 E，$K=\mathbb{F}_{2^m}$。

(iii) 式 (3.6) 的超奇异椭圆曲线 E，$K=\mathbb{F}_{2^m}$。



(a) 相加：$P+Q=R$　　(b) 倍点：$P+P=R$

图 3.3　椭圆曲线的点加和倍点运算的几何表示

群的运算法则（E/K：$y^2=x^3+ax+b$，域 K 的特征不等于 2 或 3）

1. 单位元。对于所有的 $P \in E(K)$，$P+\infty=\infty+P=P$。

2. 负元素。若 $P=(x,y)\in E(K)$，则$(x,y)+(x,-y)=\infty$。记点$(x,-y)$为$-P$，并称其为 P 的负。注意，$-P$ 也是 $E(K)$上的一个点，此外 $-\infty=\infty$。
3. 点加。令 $P=(x_1,y_1)\in E(K)$，$Q=(x_2,y_2)\in E(K)$，$P\neq\pm Q$，则$P+Q=(x_3,y_3)$。其中，

$$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2 \text{ 和 } y_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1。$$

4. 倍点。令$P=(x_1,y_1)\in E(K)$，$P\neq -P$，则 $2P=(x_3,y_3)$。其中，

$$x_3=\left(\frac{3x_1^2+a}{2y_1}\right)^2-2x_1 \text{ 和 } y_3=\left(\frac{3x_1^2+a}{2y_1}\right)(x_1-x_3)-y_1。$$

例 3.5（素域$\mathbb{F}_{29}$上的椭圆曲线）令$p=29, a=4$，$b=20$，考虑定义在素域$\mathbb{F}_{29}$上的椭圆曲线

$$E: y^2=x^3+4x+20。$$

注意，$\Delta=-16(4a^3+27b^2)=-176\,896\not\equiv 0 \pmod{29}$，所以曲线 E 确实是椭圆曲线。椭圆曲线 $E(\mathbb{F}_{29})$的点如下：

∞	(2,6)	(4,19)	(8,10)	(13,23)	(16,2)	(19,16)	(27,2)
(0,7)	(2,23)	(5,7)	(8,19)	(14,6)	(16,27)	(20,3)	(27,27)
(0,22)	(3,1)	(5,22)	(10,4)	(14,23)	(17,10)	(20,26)	
(1,5)	(3,28)	(6,12)	(10,25)	(15,2)	(17,19)	(24,7)	
(1,24)	(4,10)	(6,17)	(13,6)	(15,27)	(19,13)	(24,22)	

椭圆曲线的点加例子是 (5, 22) + (16, 27) = (13, 6) 和 2(5, 22) = (14, 6)。

群的运算法则（非超奇异椭圆曲线 $E/\mathbb{F}_{2^m}: y^2+xy=x^3+ax^2+b$）

1. 单位元。对于所有的$P\in E(\mathbb{F}_{2^m})$，$P+\infty=\infty+P=P$。
2. 负元素。若 $P=(x,y)\in E(\mathbb{F}_{2^m})$，则$(x,y)+(x,x+y)=\infty$。记点$(x,x+y)$为$-P$，并称其为 P 的负。注意，$-P$ 也是$E(\mathbb{F}_{2^m})$上的一个点，此外$-\infty=\infty$。
3. 点加。令$P=(x_1,y_1)\in E(\mathbb{F}_{2^m})$，$Q=(x_2,y_2)\in E(\mathbb{F}_{2^m})$，$P\neq\pm Q$，则 $P+Q=(x_3,y_3)$。其中，

$$x_3=\lambda^2+\lambda+x_1+x_2+a \text{和} y_3=\lambda(x_1+x_3)+x_3+y_1,$$

而$\lambda=(y_1+y_2)/(x_1+x_2)$。

4. 倍点。令$P=(x_1,y_1)\in E(\mathbb{F}_{2^m})$，$P\neq -P$，则 $2P=(x_3,y_3)$。其中，

$$x_3=\lambda^2+\lambda+a=x_1^2+\frac{b}{x_1^2} \text{ 和 } y_3=x_1^2+\lambda x_3+x_3,$$

而$\lambda=x_1+y_1/x_1$。

例 3.6 （域$\mathbb{F}_{2^4}$上的非超奇异椭圆曲线）考虑由约减多项式$f(z)=z^4+z+1$（参见例 2.2）所构成的有限域$\mathbb{F}_{2^4}$。域$\mathbb{F}_{2^4}$的一个元素$a_3z^3+a_2z^2+a_1z+a_0$可表示成一个长度为 4 的字符串$(a_3a_2a_1a_0)$。例如，(0101)表示z^2+1。令$a=z^3$，$b=z^3+1$。考虑定义在域$\mathbb{F}_{2^4}$上的非超奇异椭圆曲线

$$E: y^2+xy=x^3+z^3x^2+(z^3+1)。$$

椭圆曲线$E(\mathbb{F}_{2^4})$的点如下：

∞	(0011, 1100)	(1000, 0001)	(1100, 0000)
(0000, 1011)	(0011, 1111)	(1000, 1001)	(1100, 1100)
(0001, 0000)	(0101, 0000)	(1001, 0110)	(1111, 0100)
(0001, 0001)	(0101, 0101)	(1001, 1111)	(1111, 1011)
(0010, 1101)	(0111, 1011)	(1011, 0010)	
(0010, 1111)	(0111, 1100)	(1011, 1001)	

椭圆曲线的点加例子是 (0010, 1111) + (1100, 1100) = (0001, 0001)，2(0010, 1111) = (1011, 0010)。

群的运算法则（超奇异椭圆曲线 $E/\mathbb{F}_{2^m}: y^2+cy=x^3+ax+b$）

1. 单位元。对于所有的$P\in E(\mathbb{F}_{2^m})$，$P+\infty=\infty+P=P$。
2. 负元素。若$P=(x,y)\in E(\mathbb{F}_{2^m})$，则$(x,y)+(x,y+c)=\infty$。记点$(x,y+c)$为$-P$，并称其为$P$的负。注意，$-P$也是$E(\mathbb{F}_{2^m})$上的一个点，此外$-\infty=\infty$。
3. 点加。令$P=(x_1,y_1)\in E(\mathbb{F}_{2^m}), Q=(x_2,y_2)\in E(\mathbb{F}_{2^m}), P\neq\pm Q$，则$P+Q=(x_3,y_3)$。其中，

$$x_3=\left(\frac{y_1+y_2}{x_1+x_2}\right)^2+x_1+x_2 \text{ 和 } y_3=\left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3)+y_1+c。$$

4. 倍点。令$P=(x_1,y_1)\in E(\mathbb{F}_{2^m})$，$P\neq -P$，则$2P=(x_3,y_3)$。其中，

$$x_3=\left(\frac{x_1^2+a}{c}\right)^2 \text{和} y_3=\left(\frac{x_1^2+a}{c}\right)(x_1+x_3)+y_1+c。$$

3.1.3　群的阶

设 E 是定义在域 $\mathbb{F}_q$ 上的椭圆曲线。椭圆曲线 $E(\mathbb{F}_q)$ 的点的个数记为 $\#E(\mathbb{F}_q)$，并称其为域 $\mathbb{F}_q$ 上椭圆曲线 E 的阶。因为 Weierstrass 方程 (3.1) 对于每个 $x \in \mathbb{F}_q$ 最多有两个解，所以我们有 $\#E(\mathbb{F}_q) \in [1, 2q+1]$。Hasse 定理给出了 $\#E(\mathbb{F}_q)$ 的较紧的界。

定理 3.7　（Hasse 定理）设 E 是定义在域 $\mathbb{F}_q$ 上的椭圆曲线，则

$$q+1-2\sqrt{q} \leqslant \#E(\mathbb{F}_q) \leqslant q+1+2\sqrt{q}。$$

区间 $[q+1-2\sqrt{q},\ q+1+2\sqrt{q}]$ 称为 Hasse 区间。Hasse 定理的另一种描述是，若 E 是定义在域 $\mathbb{F}_q$ 上的椭圆曲线，则 $\#E(\mathbb{F}_q)=q+1-t$，其中 t 的绝对值 $|t| \leqslant 2\sqrt{q}$，t 称为域 $\mathbb{F}_q$ 上椭圆曲线 E 的迹。因为 $2\sqrt{q}$ 小于 q，所以我们有 $\#E(\mathbb{F}_q) \approx q$。

下面的定理确定了当 E 遍历 $\mathbb{F}_q$ 上的所有椭圆曲线时 $\#E(\mathbb{F}_q)$ 的可能值。

定理 3.8　（椭圆曲线可能的阶）令 $q=p^m$，其中 p 是域 $\mathbb{F}_q$ 的特征。存在定义在域 $\mathbb{F}_q$ 上的阶为 $\#E(\mathbb{F}_q)=q+1-t$ 的椭圆曲线 E，当且仅当下列条件之一成立：

(i)　$t \not\equiv 0 \pmod p$ 且 $t^2 \leqslant 4q$。

(ii)　m 是奇数且下列条件之一成立：(a) $t=0$；(b) $t^2=2q$ 且 $p=2$；(c) $t^2=3q$ 且 $p=3$。

(iii)　m 是偶数且下列条件之一成立：(a) $t^2=4q$；(b) $t^2=q$ 且 $p \not\equiv 1 \pmod 3$；(c) $t=0$ 且 $p \not\equiv 1 \pmod 4$。

定理 3.8 的一个推论是，对于任意的素数 p 和满足 $|t| \leqslant 2\sqrt{p}$ 的整数 t，存在阶为 $\#E(\mathbb{F}_p)=p+1-t$ 的域 $\mathbb{F}_p$ 上的椭圆曲线 E。关于这一结论，我们用例 3.9 来说明。

例 3.9　（域 $\mathbb{F}_{37}$ 上的椭圆曲线的阶）令 $p=37$。对于每个在 Hasse 区间 $[37+1-2\sqrt{37},\ 37+1+2\sqrt{37}]$ 内的整数 n，表 3.1 给出了使定义在域 $\mathbb{F}_{37}$ 上的椭圆曲线 $E: y^2=x^3+ax+b$ 的阶 $\#E(\mathbb{F}_{37})=n$ 的系数 (a, b)。

表 3.1　定义在域 $\mathbb{F}_{37}$ 上的椭圆曲线 $E: y^2=x^3+ax+b$ 的可能的阶 $n=\#E(\mathbb{F}_{37})$

n	(a,b)	n	(a,b)	n	(a,b)	n	(a,b)	n	(a,b)
26	(5,0)	31	(2,8)	36	(1,0)	41	(1,16)	46	(1,11)
27	(0,9)	32	(3,6)	37	(0,5)	42	(1,9)	47	(3,15)
28	(0,6)	33	(1,13)	38	(1,5)	43	(2,9)	48	(0,1)
29	(1,12)	34	(1,18)	39	(0,3)	44	(1,7)	49	(0,2)
30	(2,2)	35	(1,8)	40	(1,2)	45	(2,14)	50	(2,0)

阶$\#E(\mathbb{F}_q)$可以用来定义一个椭圆曲线的超奇异性。

定义 3.10 令 p 是域$\mathbb{F}_q$的特征。E 是定义在域$\mathbb{F}_q$上的椭圆曲线，若 p 整除 t，其中 t 是迹，则曲线 E 是超奇异的。若 p 不能整除 t，则曲线 E 是非超奇异的。

若 E 是定义在域$\mathbb{F}_q$上的椭圆曲线，则曲线 E 也定义在$\mathbb{F}_q$的任何扩域$\mathbb{F}_{q^n}$上。域$\mathbb{F}_q$上的有理数点群$E(\mathbb{F}_q)$是域$\mathbb{F}_{q^n}$上的有理数点群$E(\mathbb{F}_{q^n})$的子群，因此$\#E(\mathbb{F}_q)$ 整除 $\#E(\mathbb{F}_{q^n})$。若知道了$\#E(\mathbb{F}_q)$，则根据下面的定理可以很快地确定出$\#E(\mathbb{F}_{q^n})$。

定理 3.11 令 E 是定义在域 $\mathbb{F}_q$ 上的椭圆曲线，并令$\#E(\mathbb{F}_q) = q+1-t$。则对于所有的 $n \geqslant 2$，$\#E(\mathbb{F}_{q^n}) = q^n + 1 - V_n$，其中$\{V_n\}$是如下递归定义的序列：$V_0 = 2, V_1 = t$，并且对于所有的 $n \geqslant 2$，都有 $V_n = V_1 V_{n-1} - qV_{n-2}$。

3.1.4 群的结构

定理 3.12 描述了群$E(\mathbb{F}_q)$的结构。我们用 $\mathbb{Z}_n$ 表示 n 阶循环群。

定理 3.12 （*椭圆曲线的群结构*）令 E 是定义在域$\mathbb{F}_q$上的椭圆曲线，则$E(\mathbb{F}_q)$与$\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$同构，其中 n_1 和 n_2 是使 n_2 同时整除 n_1 和 $q-1$ 的正整数。

注意，$\#E(\mathbb{F}_q) = n_1 n_2$。若$n_2 = 1$，则 $E(\mathbb{F}_q)$是循环群。若$n_2 > 1$，则 $E(\mathbb{F}_q)$的秩是 2。若 n_2是小整数（如 2，3 或 4），则我们有时称 $E(\mathbb{F}_q)$ 是殆循环的。因为 n_2 同时整除 n_1 和 $q-1$，所以我们期望$E(\mathbb{F}_q)$ 对于 $\mathbb{F}_q$ 上的多数椭圆曲线 E 是循环的或是殆循环的。

例 3.13 （*群的结构*）定义在素域 $\mathbb{F}_{29}$ 上的椭圆曲线 $E: y^2 = x^3 + 4x + 20$（参见例 3.5）的$\#E(\mathbb{F}_{29}) = 37$。因为 37 是素数，所以$E(\mathbb{F}_{29})$是循环群，而且$E(\mathbb{F}_{29})$中除了无穷远点 ∞外的任何一个点都是生成元。下面给出了用点 $P(1, 5)$的乘法生成的群$E(\mathbb{F}_{29})$的全体元素。

$0P = \infty$	$8P = (8, 10)$	$16P = (0, 22)$	$24P = (16, 2)$	$32P = (6, 17)$
$1P = (1, 5)$	$9P = (14, 23)$	$17P = (27, 2)$	$25P = (19, 16)$	$33P = (15, 2)$
$2P = (4, 19)$	$10P = (13, 23)$	$18P = (2, 23)$	$26P = (10, 4)$	$34P = (20, 26)$
$3P = (20, 3)$	$11P = (10, 25)$	$19P = (2, 6)$	$27P = (13, 6)$	$35P = (4, 10)$
$4P = (15, 27)$	$12P = (19, 13)$	$20P = (27, 27)$	$28P = (14, 6)$	$36P = (1, 24)$
$5P = (6, 12)$	$13P = (16, 27)$	$21P = (0, 7)$	$29P = (8, 19)$	
$6P = (17, 19)$	$14P = (5, 22)$	$22P = (3, 28)$	$30P = (24, 7)$	
$7P = (24, 22)$	$15P = (3, 1)$	$23P = (5, 7)$	$31P = (17, 10)$	

例 3.14 （*群的结构*）考虑由约减多项式 $f(z) = z^4 + z + 1$所构成的有限域 $\mathbb{F}_{2^4}$。定义在域 $\mathbb{F}_{2^4}$

上的椭圆曲线$E: y^2+xy=x^3+z^3x^2+(z^3+1)$（参见例 3.6）的$\#E(\mathbb{F}_{2^4})=22$。因为 22 没有重因子，所以$E(\mathbb{F}_{2^4})$是循环群。点$P=(z^3,1)=(1000,0001)$的阶为 11。下面给出了点 P 的乘法运算。

$0P=\infty$	$3P=(1100,0000)$	$6P=(1011,1001)$	$9P=(1001,0110)$
$1P=(1000,0001)$	$4P=(1111,1011)$	$7P=(1111,0100)$	$10P=(1000,1001)$
$2P=(1001,1111)$	$5P=(1011,0010)$	$8P=(1100,1100)$	

3.1.5　同构类

回忆椭圆曲线同构的定义（定义 3.4）。同构关系是定义在有限域 K 上的椭圆曲线集合上的等价关系。若椭圆曲线 E_1 和 E_2 在 K 上是同构的，则它们的有理数点群 $E_1(K)$ 和 $E_2(K)$ 也是同构的。然而，逆命题不成立（见例 3.16 和例 3.17）。下面我们针对定义在特征不等于 2 或 3 的有限域上的椭圆曲线和定义在二进制有限域上的非超奇异椭圆曲线两种情况，给出关于椭圆曲线同构类的一些结论。

定理 3.15　（椭圆曲线的同构类）令 $K=\mathbb{F}_q$ 是一个特征不等于 2 或 3 的有限域。

(i) 定义在 K 上的椭圆曲线

$$E_1: y^2=x^3+ax+b \tag{3.9}$$

$$E_2: y^2=x^3+\bar{a}x+\bar{b} \tag{3.10}$$

是同构的，当且仅当存在一个 $u\in K^*$，使得 $u^4\bar{a}=a$ 且 $u^6\bar{b}=b$。若这样的 u 存在，则变量的相容性变换

$$(x,y)\to(u^2x,u^3y)$$

把式 (3.9) 变换为式 (3.10)。

(ii) 对于 $q\equiv 1,5,7,11 \pmod{12}$，定义在有限域 K 上的椭圆曲线的同构类的数量分别是 $2q+6$, $2q+2$, $2q+4$, $2q$。

例 3.16　（定义在有限域 $\mathbb{F}_5$ 上的椭圆曲线的同构类）表 3.2 列出了 12 个定义在有限域 $\mathbb{F}_5$ 上的椭圆曲线的同构类。注意，若域 $\mathbb{F}_q$ 的有理数点群 $E_1(\mathbb{F}_q)$ 和 $E_2(\mathbb{F}_q)$ 是同构的，则这并不意味着椭圆曲线 E_1 和 E_2 在 $\mathbb{F}_q$ 上也是同构的。例如，椭圆曲线 $E_1: y^2=x^3+1$ 和 $E_2: y^2=x^3+2$ 在 $\mathbb{F}_5$ 上不同构，但是 $E_1(\mathbb{F}_5)$ 和 $E_2(\mathbb{F}_5)$ 的阶都是 6，因此这两个群都与 $\mathbb{Z}_6$ 同构。

表 3.2 $\mathbb{F}_5$上椭圆曲线 E 的同构类

同构类	$\#E(\mathbb{F}_5)$	$E(\mathbb{F}_5)$的群结构
$\{y^2=x^3+1,\ y^2=x^3+4\}$	6	$\mathbb{Z}_6$
$\{y^2=x^3+2,\ y^2=x^3+3\}$	6	$\mathbb{Z}_6$
$\{y^3=x^3+x\}$	4	$\mathbb{Z}_2\oplus\mathbb{Z}_2$
$\{y^3=x^3+2x\}$	2	$\mathbb{Z}_2$
$\{y^3=x^3+3x\}$	10	$\mathbb{Z}_{10}$
$\{y^3=x^3+4x\}$	8	$\mathbb{Z}_4\oplus\mathbb{Z}_2$
$\{y^2=x^3+x+1,\ y^2=x^3+x+4\}$	9	$\mathbb{Z}_9$
$\{y^2=x^3+x+2,\ y^2=x^3+x+3\}$	4	$\mathbb{Z}_4$
$\{y^2=x^3+2x+1,\ y^2=x^3+2x+4\}$	7	$\mathbb{Z}_7$
$\{y^2=x^3+3x+2,\ y^2=x^3+3x+3\}$	5	$\mathbb{Z}_5$
$\{y^2=x^3+4x+1,\ y^2=x^3+4x+4\}$	8	$\mathbb{Z}_8$
$\{y^2=x^3+4x+2,\ y^2=x^3+4x+3\}$	3	$\mathbb{Z}_3$

例 3.17 令 $p=73$，利用定理 3.15 容易验证定义在$\mathbb{F}_p$上的椭圆曲线

$$E_1: y^2=x^3+25x$$
$$E_2: y^2=x^3+53x+55$$

在$\mathbb{F}_p$上不同构。然而，域$\mathbb{F}_{p^m}$上的有理数点群$E_1(\mathbb{F}_{p^m})$和$E_2(\mathbb{F}_{p^m})$却对于任意的 $m\geqslant 1$ 都同构。

定理 3.18 （二进制域上椭圆曲线的同构类）令 $K=\mathbb{F}_{2^m}$是一个二进制域。

(i) 定义在 K 上的非超奇异椭圆曲线

$$E_1: y^2+xy=x^3+ax^2+b \tag{3.11}$$
$$E_2: y^2+xy=x^3+\bar{a}x^2+\bar{b} \tag{3.12}$$

在 K 上同构，当且仅当$b=\bar{b}$且$\mathrm{Tr}(a)=\mathrm{Tr}(\bar{a})$，其中 Tr 是迹函数（见定义 3.78）。若这些条件满足，则存在$s\in\mathbb{F}_{2^m}$使得$\bar{a}=s^2+s+a$，并且存在变量的相容性变换

$$(x,y)\to(x,y+sx)$$

把式 (3.11) 变换成式 (3.12)。

(ii) K 上的非超奇异椭圆曲线的同构类的数量为$2^{m+1}-2$。令$\gamma\in\mathbb{F}_{2^m}$且满足$\mathrm{Tr}(\gamma)=1$，则同构类的代表集合是

$$\{y^2+xy=x^3+ax^2+b \mid a\in\{0,\gamma\},\ b\in\mathbb{F}_{2^m}^*\}。$$

(iii) 非超奇异椭圆曲线 $E: y^2+xy=x^3+\gamma x^2+b$ 的阶 $\#E(\mathbb{F}_{2^m})$ 能被 2 整除。进而，若 $\mathrm{Tr}(\gamma)=0$，则 $\#E(\mathbb{F}_{2^m})$能被 4 整除。

3.2　点的表示与群的运算法则

设域 K 的特征为 2 或 3，我们在 3.1 节给出了定义在域 K 上的椭圆曲线 $y^2=x^3+ax+b$ 的两个点的加法公式，还给出了二进制域上椭圆曲线 $y^2+xy=x^3+ax^2+b$ 的两个点的加法公式。对于这两类曲线，点加（两个互不相同且互不为负的点相加）和倍点运算需要用到域上的求逆和乘法。若域 K 上的求逆比乘法费时，则采用投影坐标来表示点将是非常好的一种解决办法。

3.2.1　投影坐标

令 K 是一个域，令 c 和 d 是正整数，集合 $K^3\backslash\{(0,0,0)\}$ 是域 K 上非零三维向量的集合。我们可以定义一个集合 $K^3\backslash\{(0,0,0)\}$ 上的等价关系~如下：

$(X_1,Y_1,Z_1)\sim(X_2,Y_2,Z_2)$，若对于某个 $\lambda\in K^*$ 有 $X_1=\lambda^c X_2, Y_1=\lambda^d Y_2, Z_1=\lambda Z_2$。

包含 $(X,Y,Z)\in K^3\backslash\{(0,0,0)\}$ 的等价类是

$$(X:Y:Z)=\{(\lambda^c X,\lambda^d Y,\lambda Z):\lambda\in K^*\}。$$

$(X:Y:Z)$称为投影点，(X,Y,Z)称为 $(X:Y:Z)$的代表元。所有投影点的集合用 $\mathbb{P}(K)$表示。注意，若 $(X',Y',Z')\in(X:Y:Z)$，则 $(X':Y':Z')=(X:Y:Z)$，即等价类的任一元素都可以作为它的代表元。实际上，若 $Z\neq 0$，则 $(X/Z^c,Y/Z^d,1)$ 是投影点 $(X:Y:Z)$的代表元，并且是 Z 坐标为 1 的惟一的代表元。于是，我们有投影点集合

$$\mathbb{P}(K)^*=\{(X:Y:Z):X,Y,Z\in K,Z\neq 0\}$$

和仿射点集合

$$\mathbb{A}(K)=\{(x,y):x,y\in K\}$$

之间的一一映射。

投影点集合

$$\mathbb{P}(K)^0=\{(X:Y:Z):X,Y,Z\in K,Z=0\}$$

称为无穷远的线，因为它的点并不对应于任何仿射点。通过用X/Z^c代替x和用Y/Z^d代替y，并去掉分母，可以得到定义在K上的椭圆曲线E的 Weierstrass 方程 (3.1) 的投影坐标形式。现在，若$(X,Y,Z)\in K^3\backslash\{(0,0,0)\}$满足投影方程，则任何$(X',Y',Z')\in(X:Y:Z)$。所以，可以说投影点 $(X:Y:Z)$在曲线E上。因此，$\mathbb{A}(K)$中的仿射点和$\mathbb{P}(K)^*$中的投影点之间有一一映射关系。$\mathbb{P}(K)^0$中的投影点是E的无穷远点。

例 3.19 （标准投影坐标）令$c=1,d=1$。则定义在K上的 Weierstrass 方程

$$E: y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$$

的投影形式是

$$Y^2Z+a_1XYZ+a_3YZ^2=X^3+a_2X^2Z+a_4XZ^2+a_6Z^3。$$

曲线E上惟一的一个无穷远点是$(0:1:0)$。这个投影点对应于定义 3.1 中的点∞。

投影坐标的点加和倍点运算不涉及域上的求逆。首先把点转换为仿射坐标，然后利用 3.1 节的公式进行仿射点加，最后去掉分母，便得到投影坐标点的计算公式。另一种应用于点乘的方法（见 3.3 节）是两个混合坐标的点加，这两个点分别用不同的坐标表示。

例 3.20 （利用雅可比坐标的加法公式）令$c=2,d=3$，投影点$(X:Y:Z)$，$Z\neq0$与仿射点$(X/Z^2,Y/Z^3)$对应。则定义在K上的 Weierstrass 方程

$$E: y^2=x^3+ax+b$$

的投影形式是

$$Y^2=X^3+aXZ^4+bZ^6。$$

无穷远点与点$(1:1:0)$对应，而点$(X:Y:Z)$的负点是$(X:-Y:Z)$。

倍点。令$P=(X_1:Y_1:Z_1)\in E$，并假设$P\neq-P$。因为$P=(X_1/Z_1^2:Y_1/Z_1^3:1)$，所以我们可以利用仿射坐标形式的倍点公式计算$2P=(X_3':Y_3':1)$，得到

$$X_3'=\left(\frac{3\frac{X_1^2}{Z_1^4}+a}{2\frac{Y_1}{Z_1^3}}\right)^2-2\frac{X_1}{Z_1^2}=\frac{(3X_1^2+aZ_1^4)^2-8X_1Y_1^2}{4Y_1^2Z_1^2}$$

和

$$Y_3' = \left(\frac{3\frac{X_1^2}{Z_1^4}+a}{2\frac{Y_1}{Z_1^3}}\right)\left(\frac{X_1}{Z_1^2}-X_3'\right)-\frac{Y_1}{Z_1^3} = \frac{3X_1^2+aZ_1^4}{2Y_1Z_1}\left(\frac{X_1}{Z_1^2}-X_3'\right)-\frac{Y_1}{Z_1^3}。$$

为了在 X_3' 和 Y_3' 的表达式中消去分母，我们令 $X_3 = X_3' \cdot Z_3^2$，$Y_3 = Y_3' \cdot Z_3^3$，其中 $Z_3 = 2Y_1Z_1$，于是得到雅可比坐标形式的计算 $2P = (X_3 : Y_3 : Z_3)$ 的公式：

$$\left.\begin{aligned} X_3 &= (3X_1^2+aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 &= (3X_1^2+aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned}\right\} \tag{3.13}$$

通过存储某些中间结果，可通过如下 6 次域的平方和 4 次域的乘法计算出 X_3, Y_3 和 Z_3：

$$A \leftarrow Y_1^2,\quad B \leftarrow 4X_1 \cdot A,\quad C \leftarrow 8A^2,\quad D \leftarrow 3X_1^2 + a \cdot Z_1^4,$$
$$X_3 \leftarrow D^2 - 2B,\quad Y_3 \leftarrow D \cdot (B - X_3) - C,\quad Z_3 \leftarrow 2Y_1 \cdot Z_1。$$

使用雅可比和仿射混合坐标的点加。令 $P = (X_1 : Y_1 : Z_1) \in E$, $Z_1 \neq 0$, $Q = (X_2 : Y_2 : 1)$，并且假设 $P \neq \pm Q$。因为 $P = (X_1/Z_1^2 : Y_1/Z_1^3 : 1)$，所以我们可以用仿射坐标中的加法公式计算 $P + Q = (X_3' : Y_3' : 1)$，并得到

$$X_3' = \left(\frac{Y_2 - \frac{Y_1}{Z_1^3}}{X_2 - \frac{X_1}{Z_1^2}}\right)^2 - \frac{X_1}{Z_1^2} - X_2 = \left(\frac{Y_2Z_1^3 - Y_1}{(X_2Z_1^2 - X_1)Z_1}\right)^2 - \frac{X_1}{Z_1^2} - X_2$$

和

$$Y_3' = \left(\frac{Y_2 - \frac{Y_1}{Z_1^3}}{X_2 - \frac{X_1}{Z_1^2}}\right)\left(\frac{X_1}{Z_1^2} - X_3'\right) - \frac{Y_1}{Z_1^3} = \left(\frac{Y_2Z_1^3 - Y_1}{(X_2Z_1^2 - X_1)Z_1}\right)\left(\frac{X_1}{Z_1^2} - X_3'\right) - \frac{Y_1}{Z_1^3}。$$

为了在 X_3' 和 Y_3' 的表达式中消去分母，我们令 $X_3 = X_3' \cdot Z_3^2$, $Y_3 = Y_3' \cdot Z_3^3$, 其中 $Z_3 = (X_2Z_1^2 - X_1)Z_1$，于是得到计算 $P + Q = (X_3 : Y_3 : Z_3)$ 的雅可比坐标的公式：

$$\left.\begin{aligned} X_3 &= (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)^2(X_1 + X_2Z_1^2) \\ Y_3 &= (Y_2Z_1^3 - Y_1)(X_1(X_2Z_1^2 - X_1)^2 - X_3) - Y_1(X_2Z_1^2 - X_1)^3 \\ Z_3 &= (X_2Z_1^2 - X_1)Z_1 \end{aligned}\right\} \tag{3.14}$$

通过存储某些中间结果，可通过如下 3 次域的平方和 8 次域的乘法计算出 X_3, Y_3 和 Z_3：

$$A \leftarrow Z_1^2,\quad B \leftarrow Z_1 \cdot A,\quad C \leftarrow X_2 \cdot A,\quad D \leftarrow Y_2 \cdot B,\quad E \leftarrow C - X_1,$$
$$F \leftarrow D - Y_1,\quad G \leftarrow E^2,\quad H \leftarrow G \cdot E,\quad I \leftarrow X_1 \cdot G,$$
$$X_3 \leftarrow F^2 - (H + 2I),\quad Y_3 \leftarrow F \cdot (I - X_3) - Y_1 \cdot H,\quad Z_3 \leftarrow Z_1 \cdot E。$$

3.2.2 椭圆曲线 $y^2 = x^3 + ax + b$

本节讨论定义在域 K（特征不等于 2 且不等于 3）上的椭圆曲线 $E: y^2 = x^3 + ax + b$ 的坐标系统和加法公式。在这里引入了几种投影坐标。

1. 标准投影坐标。这里 $c = 1$ 且 $d = 1$。投影点 $(X : Y : Z)$, $Z \neq 0$ 与仿射点 $(X/Z, Y/Z)$ 对应。则椭圆曲线的投影方程是

$$Y^2 Z = X^3 + aXZ^2 + bZ^3。$$

无穷远点 ∞ 与点 $(0:1:0)$ 对应，而点 $(X : Y : Z)$ 的负点是 $(X : -Y : Z)$。

2. 雅可比投影坐标。这里 $c = 2$ 且 $d = 3$。投影点 $(X : Y : Z)$, $Z \neq 0$ 与仿射点 $(X/Z^2, Y/Z^3)$ 对应。则椭圆曲线的投影方程是

$$Y^2 = X^3 + aXZ^4 + bZ^6。$$

无穷远点 ∞ 与点 $(1 : 1 : 0)$ 对应，而点 $(X : Y : Z)$ 的负点是 $(X : -Y : Z)$。倍点和点加的公式在例 3.20 中已给出。若 $a = -3$，则通过 1 次域乘法和 1 次平方运算就可计算出式(3.13)中的表达式 $3X_1^2 + aZ_1^4$，因为

$$3X_1^2 - 3Z_1^4 = 3(X_1 - Z_1^2) \cdot (X_1 + Z_1^2)。$$

自此以后，我们将假设椭圆曲线 $y^2 = x^3 + ax + b$ 中的 $a = -3$。定理 3.15 认定这种假设不失一般性。利用 $2Y_1$ 在式 (3.13) 中出现了几次的事实，并用乘 4 和乘 8 的计算替换除 2 的计算，则倍点的计算将会提高速度。修改后的倍点公式是

$$A \leftarrow 3(X_1 - Z_1^2) \cdot (X_1 + Z_1^2),\quad B \leftarrow 2Y_1,\quad Z_3 \leftarrow B \cdot Z_1,\quad C \leftarrow B^2,$$
$$D \leftarrow C \cdot X_1,\quad X_3 \leftarrow A^2 - 2D,\quad Y_3 \leftarrow (D - X_3) \cdot A - C^2/2。$$

$a = -3$ 时的点加和倍点计算过程在算法 3.21 和算法 3.22 中给出，其中努力使临时变量 T_i 的数最小。算法用域的基本运算来书写，然而用基本运算集成的专用程序将是更优的（见 5.1.2 节，在那里给出了利用浮点运算硬件进行快速计算的具体例子）。

3. Chudnovsky 坐标。这里雅可比点 $(X : Y : Z)$ 被表示为 $(X : Y : Z : Z^2 : Z^3)$。这种表示的冗余性在某些采用投影坐标的点乘方法中是有益的。

算法 3.21　倍点（$y^2=x^3-3x+b$，雅可比坐标）

输入：椭圆曲线 $E/K: y^2=x^3-3x+b$，雅可比坐标中的点 $P=(X_1:Y_1:Z_1)$。

输出：雅可比坐标中的点 $2P=(X_3:Y_3:Z_3)$。

1. 若 $P=\infty$，则返回(∞)。
2. $T_1 \leftarrow Z_1^2$。　$\{T_1 \leftarrow Z_1^2\}$
3. $T_2 \leftarrow X_1-T_1$。　$\{T_2 \leftarrow X_1-Z_1^2\}$
4. $T_1 \leftarrow X_1+T_1$。　$\{T_1 \leftarrow X_1+Z_1^2\}$
5. $T_2 \leftarrow T_2 \cdot T_1$。　$\{T_2 \leftarrow X_1^2-Z_1^4\}$
6. $T_2 \leftarrow 3T_2$。　$\{T_2 \leftarrow A=3(X_1-Z_1^2)(X_1+Z_1^2)\}$
7. $Y_3 \leftarrow 2Y_1$。　$\{Y_3 \leftarrow B=2Y_1\}$
8. $Z_3 \leftarrow Y_3 \cdot Z_1$。　$\{Z_3 \leftarrow BZ_1\}$
9. $Y_3 \leftarrow Y_3^2$。　$\{Y_3 \leftarrow C=B^2\}$
10. $T_3 \leftarrow Y_3 \cdot X_1$。　$\{T_3 \leftarrow D=CX_1\}$
11. $Y_3 \leftarrow Y_3^2$。　$\{Y_3 \leftarrow C^2\}$
12. $Y_3 \leftarrow Y_3/2$。　$\{Y_3 \leftarrow C^2/2\}$
13. $X_3 \leftarrow T_2^2$。　$\{X_3 \leftarrow A^2\}$
14. $T_1 \leftarrow 2T_3$。　$\{T_1 \leftarrow 2D\}$
15. $X_3 \leftarrow X_3-T_1$。　$\{X_3 \leftarrow A^2-2D\}$
16. $T_1 \leftarrow T_3-X_3$。　$\{T_1 \leftarrow D-X_3\}$
17. $T_1 \leftarrow T_1 \cdot T_2$。　$\{T_1 \leftarrow (D-X_3)A\}$
18. $Y_3 \leftarrow T_1-Y_3$。　$\{Y_3 \leftarrow (D-X_3)A-C^2/2\}$
19. 返回$(X_3:Y_3:Z_3)$。

算法 3.22　点加（$y^2=x^3-3x+b$，仿射-雅可比坐标）

输入：椭圆曲线 $E/K: y^2=x^3-3x+b$，雅可比坐标中的点 $P=(X_1:Y_1:Z_1)$，仿射坐标中的点 $Q=(x_2,y_2)$。

输出：雅可比坐标中的 $P+Q=(X_3:Y_3:Z_3)$。

1. 若 $Q=\infty$，则返回$(X_1:Y_1:Z_1)$。
2. 若 $P=\infty$，则返回$(x_2:y_2:1)$。
3. $T_1 \leftarrow Z_1^2$。　$\{T_1 \leftarrow A=Z_1^2\}$
4. $T_2 \leftarrow T_1 \cdot Z_1$。　$\{T_2 \leftarrow B=Z_1A\}$
5. $T_1 \leftarrow T_1 \cdot x_2$。　$\{T_1 \leftarrow C=X_2A\}$
6. $T_2 \leftarrow T_2 \cdot y_2$。　$\{T_2 \leftarrow D=Y_2B\}$
7. $T_1 \leftarrow T_1-X_1$。　$\{T_1 \leftarrow E=C-X_1\}$
8. $T_2 \leftarrow T_2-Y_1$。　$\{T_2 \leftarrow F=D-Y_1\}$

9. 若$T_1=0$，则

9.1 若$T_2=0$，则用算法 3.21 计算$(X_3:Y_3:Z_3)=2(x_2:y_2:1)$，并且返回$(X_3:Y_3:Z_3)$。

9.2 否则，返回(∞)。

10. $Z_3\leftarrow Z_1\cdot T_1$。 $\{Z_3\leftarrow Z_1E\}$

11. $T_3\leftarrow T_1^2$。 $\{T_3\leftarrow G=E^2\}$

12. $T_4\leftarrow T_3\cdot T_1$。 $\{T_4\leftarrow H=E^3\}$

13. $T_3\leftarrow T_3\cdot X_1$。 $\{T_3\leftarrow I=X_1G\}$

14. $T_1\leftarrow 2T_3$。 $\{T_1\leftarrow 2I\}$

15. $X_3\leftarrow T_2^2$。 $\{X_3\leftarrow F^2\}$

16. $X_3\leftarrow X_3-T_1$。 $\{X_3\leftarrow F^2-2I\}$

17. $X_3\leftarrow X_3-T_4$。 $\{X_3\leftarrow F^2-(H+2I)\}$

18. $T_3\leftarrow T_3-X_3$。 $\{T_3\leftarrow I-X_3\}$

19. $T_3\leftarrow T_3\cdot T_2$。 $\{T_3\leftarrow F(I-X_3)\}$

20. $T_4\leftarrow T_4\cdot Y_1$。 $\{T_4\leftarrow Y_1H\}$

21. $Y_3\leftarrow T_3-T_4$。 $\{Y_3\leftarrow F(I-X_3)-Y_1H\}$

22. 返回$(X_3:Y_3:Z_3)$。

在不同坐标系中点加和倍点所需的域运算的量列于表 3.3 中。其中符号$C_1+C_2\rightarrow C_3$说明要加的点在C_1坐标和C_2坐标中，而和在C_3坐标中。例如，$J+A\rightarrow J$表示雅可比坐标与仿射坐标的点加，而结果是雅可比坐标。我们看到雅可比坐标的倍点计算速度最快，而雅可比与仿射混合坐标的点加计算最快。此外还有一些点乘算法（见注 3.43）采用雅可比-Chudnovsky 混合坐标，也有一些点加算法采用 Chudnovsky-仿射混合坐标。

表 3.3 椭圆曲线$y^2=x^3-3x+b$的点加和倍点的运算量。其中 *A* 表示仿射坐标，*P* 表示标准投影坐标，*J* 表示雅可比坐标，*C* 表示 Chudnovsky 坐标，*I* 表示求逆，*M* 表示点乘，*S* 表示平方

倍 点		普通点加		混合点加	
$2A\rightarrow A$	$1I,2M,2S$	$A+A\rightarrow A$	$1I,2M,1S$	$J+A\rightarrow J$	$8M,3S$
$2P\rightarrow P$	$7M,3S$	$P+P\rightarrow P$	$12M,2S$	$J+C\rightarrow J$	$11M,3S$
$2J\rightarrow J$	$4M,4S$	$J+J\rightarrow J$	$12M,4S$	$C+A\rightarrow C$	$8M,3S$
$2C\rightarrow C$	$5M,4S$	$C+C\rightarrow C$	$11M,3S$		

连续倍点

若要进行连续倍点的运算，则算法 3.23 比重复利用倍点公式计算要快得多。从计算 $2Y$ 开始直到最后一步，只需要进行一次除 2 的计算。通过 $3(X^2-W)$来计算$3(X-Z^2)(X+Z^2)$可使得循环中的域加法被省略，其中 $W=Z^4$ 的计算在最开始的倍点计算中就完成了，之后的每一次倍点计算之前都随着 $W \leftarrow WY^4$ 的执行而被更新。

算法 3.23　连续倍点（$y^2=x^3-3x+b$，雅可比坐标）

输入：椭圆曲线 $E/K: y^2=x^3-3x+b$，雅可比坐标中的点 $P=(X:Y:Z)$，整数 $m>0$。

输出：雅可比坐标中的 $2^m P$。

1. 若 $P=\infty$，则返回(P)。
2. $Y \leftarrow 2Y, W \leftarrow Z^4$。
3. 当 $m>0$ 时，重复执行
 3.1 $A \leftarrow 3(X^2-W), B \leftarrow XY^2$。
 3.2 $X \leftarrow A^2-2B, Z \leftarrow ZY$。
 3.3 $m \leftarrow m-1$。若 $m>0$，则 $W \leftarrow WY^4$。
 3.4 $Y \leftarrow 2A(B-X)-Y^4$。
4. 返回 $(X, Y/2, Z)$。

在 m 次连续倍点计算中，算法 3.23 进行了 $m-1$ 次域的加法，$m-1$ 次用 2 除，1 次乘法和 2 次平方（与重复调用算法 3.21 比较）。这一策略可用于 $a \neq -3$ 的情况，其时在每 $m-1$ 次倍点计算中都可节省两次域平方运算。

3.2.3　椭圆曲线 $y^2+xy=x^3+ax^2+b$

本节讨论定义在二进制域 K 上的非超奇异椭圆曲线 E: $y^2+xy=x^3+ax^2+b$ 的坐标系和点加公式。其中，使用了几种不同的坐标系。

1. 标准投影坐标。这里 $c=1, d=1$。投影点 $(X:Y:Z)$, $Z \neq 0$与仿射点$(X/Z, Y/Z)$相对应。椭圆曲线的投影方程是

$$Y^2Z+XYZ=X^3+aX^2Z+bZ^3。$$

无穷远点 ∞对应于$(0:1:0)$，而点$(X:Y:Z)$的负点是$(X:X+Y:Z)$。

2. 雅可比投影坐标。这里 $c=2$, $d=3$。投影点 $(X:Y:Z)$, $Z\neq 0$与仿射点 $(X/Z^2, Y/Z^3)$相对应。椭圆曲线的投影方程是

$$Y^2+XYZ=X^3+aX^2Z^2+bZ^6。$$

无穷远点 ∞对应于$(1:1:0)$，而点$(X:Y:Z)$的负点是$(X:X+Y:Z)$。

3. López-Dahab(LD)投影坐标。这里 $c=1$, $d=2$。投影点 $(X:Y:Z)$, $Z\neq 0$与仿射点 $(X/Z, Y/Z^2)$ 相对应。椭圆曲线的投影方程是

$$Y^2+XYZ=X^3Z+aX^2Z^2+bZ^4。$$

无穷远点 ∞对应于$(1:0:0)$，而点$(X:Y:Z)$的负点是$(X:X+Y:Z)$。计算点$(X_1:Y_1:Z_1)$的倍点$(X_3:Y_3:Z_3)$的公式是

$$Z_3\leftarrow X_1^2\cdot Z_1^2,\ X_3\leftarrow X_1^4+b\cdot Z_1^4,\ Y_3\leftarrow bZ_1^4\cdot Z_3+X_3\cdot(aZ_3+Y_1^2+bZ_1^4)。$$

计算点$(X_1:Y_1:Z_1)$与点$(X_2:Y_2:1)$的和$(X_3:Y_3:Z_3)$的公式是

$$A\leftarrow Y_2\cdot Z_1^2+Y_1,\ B\leftarrow X_2\cdot Z_1+X_1,\ C\leftarrow Z_1\cdot B,\ D\leftarrow B^2\cdot(C+aZ_1^2),$$
$$Z_3\leftarrow C^2,\ E\leftarrow A\cdot C,\ X_3\leftarrow A^2+D+E,\ F\leftarrow X_3+X_2\cdot Z_3,$$
$$G\leftarrow(X_2+Y_2)\cdot Z_3^2,\ Y_3\leftarrow(E+Z_3)\cdot F+G。$$

当$a\in\{0,1\}$时，点加和倍点的计算过程由算法 3.24 和算法 3.25 给出。其中努力使临时变量T_i的数最小。定理 3.18(ii) 确认限制$a\in\{0,1\}$不失一般性。

算法 3.24 倍点（$(y^2+xy=x^3+ax^2+b, a\in\{0,1\}$，LD坐标）

输入：椭圆曲线$E/K: y^2+xy=x^3+ax^2+b$，LD坐标中的点$P=(X_1:Y_1:Z_1)$。

输出：LD坐标中的点$2P=(X_3:Y_3:Z_3)$。

1. 若$P=\infty$，则返回(∞)。
2. $T_1\leftarrow Z_1^2$。 $\{T_1\leftarrow Z_1^2\}$
3. $T_2\leftarrow X_1^2$。 $\{T_2\leftarrow X_1^2\}$
4. $Z_3\leftarrow T_1\cdot T_2$。 $\{Z_3\leftarrow X_1^2Z_1^2\}$
5. $X_3\leftarrow T_2^2$。 $\{X_3\leftarrow X_1^4\}$
6. $T_1\leftarrow T_1^2$。 $\{T_1\leftarrow Z_1^4\}$
7. $T_2\leftarrow T_1\cdot b$。 $\{T_2\leftarrow bZ_1^4\}$
8. $X_3\leftarrow X_3+T_2$。 $\{X_3\leftarrow X_1^4+bZ_1^4\}$

9. $T_1 \leftarrow Y_1^2$。　　$\{T_1 \leftarrow Y_1^2\}$
10. 若 $a=1$，则 $T_1 \leftarrow T_1 + Z_3$。　　$\{T_1 \leftarrow aZ_3 + Y_1^2\}$
11. $T_1 \leftarrow T_1 + T_2$。　　$\{T_1 \leftarrow aZ_3 + Y_1^2 + bZ_1^4\}$
12. $Y_3 \leftarrow X_3 \cdot T_1$。　　$\{Y_3 \leftarrow X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
13. $T_1 \leftarrow T_2 \cdot Z_3$。　　$\{T_1 \leftarrow bZ_1^4 Z_3\}$
14. $Y_3 \leftarrow Y_3 + T_1$。　　$\{Y_3 \leftarrow bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
15. 返回$(X_3 : Y_3 : Z_3)$。

算法 3.25　点加（$(y^2+xy=x^3+ax^2+b, a\in\{0,1\}$，LD 仿射坐标）

输入：椭圆曲线 $E/K : y^2 + xy = x^3 + ax^2 + b$，LD 坐标中的点 $P = (X_1 : Y_1 : Z_1)$，仿射坐标中的点 $Q = (x_2, y_2)$。

输出：LD坐标中的点 $P + Q = (X_3 : Y_3 : Z_3)$。

1. 若 $Q = \infty$，则返回(P)。
2. 若 $P = \infty$，则返回$(x_2 : y_2 : 1)$。
3. $T_1 \leftarrow Z_1 \cdot x_2$。　　$\{T_1 \leftarrow X_2 Z_1\}$
4. $T_2 \leftarrow Z_1^2$。　　$\{T_2 \leftarrow Z_1^2\}$
5. $X_3 \leftarrow X_1 + T_1$。　　$\{X_3 \leftarrow B = X_2 Z_1 + X_1\}$
6. $T_1 \leftarrow Z_1 \cdot X_3$。　　$\{T_1 \leftarrow C = Z_1 B\}$
7. $T_3 \leftarrow T_2 \cdot y_2$。　　$\{T_3 \leftarrow Y_2 Z_1^2\}$
8. $Y_3 \leftarrow Y_1 + T_3$。　　$\{Y_3 \leftarrow A = Y_2 Z_1^2 + Y_1\}$
9. 若 $X_3 = 0$，则
 9.1 若 $Y_3 = 0$，则调用算法 3.24 计算$(X_3 : Y_3 : Z_3) = 2(x_2 : y_2 : 1)$并返回$(X_3 : Y_3 : Z_3)$。
 9.2 否则，返回 (∞)。
10. $Z_3 \leftarrow T_1^2$。　　$\{Z_3 \leftarrow C^2\}$
11. $T_3 \leftarrow T_1 \cdot Y_3$。　　$\{T_3 \leftarrow E = AC\}$
12. 若 $a = 1$，则 $T_1 \leftarrow T_1 + T_2$。　　$\{T_1 \leftarrow C + aZ_1^2\}$
13. $T_2 \leftarrow X_3^2$。　　$\{T_2 \leftarrow B^2\}$
14. $X_3 \leftarrow T_2 \cdot T_1$。　　$\{X_3 \leftarrow D = B^2(C + aZ_1^2)\}$
15. $T_2 \leftarrow Y_3^2$。　　$\{T_2 \leftarrow A^2\}$
16. $X_3 \leftarrow X_3 + T_2$。　　$\{X_3 \leftarrow A^2 + D\}$

17. $X_3 \leftarrow X_3 + T_3$。 $\{X_3 \leftarrow A^2 + D + E\}$
18. $T_2 \leftarrow x_2 \cdot Z_3$。 $\{T_2 \leftarrow X_2 Z_3\}$
19. $T_2 \leftarrow T_2 + X_3$。 $\{T_2 \leftarrow F = X_3 + X_2 Z_3\}$
20. $T_1 \leftarrow Z_3^2$。 $\{T_1 \leftarrow Z_3^2\}$
21. $T_3 \leftarrow T_3 + Z_3$。 $\{T_3 \leftarrow E + Z_3\}$
22. $Y_3 \leftarrow T_3 \cdot T_2$。 $\{Y_3 \leftarrow (E + Z_3)F\}$
23. $T_2 \leftarrow x_2 + y_2$。 $\{T_2 \leftarrow X_2 + Y_2\}$
24. $T_3 \leftarrow T_1 \cdot T_2$。 $\{T_3 \leftarrow G = (X_2 + Y_2)Z_3^2\}$
25. $Y_3 \leftarrow Y_3 + T_3$。 $\{Y_3 \leftarrow (E + Z_3)F + G\}$
26. 返回$(X_3 : Y_3 : Z_3)$。

几种坐标系的点加和倍点域运算的量如表 3.4 所示。

表 3.4 椭圆曲线 $y^2 + xy = x^3 + ax^2 + b$的点加和倍点的运算量。$M$ 表示乘法，V 表示除法（见 2.3.6 节）

坐标系	普通点加	普通点加（混合坐标）	倍点
仿射	$V+M$	—	$V+M$
标准投影	13 M	12 M	7 M
雅可比投影	14 M	10 M	5 M
López-Dahab 投影	14 M	8 M	4 M

3.3 点乘

本节讨论计算 kP 的方法，其中 k 是一个整数，P 是定义在域 $\mathbb{F}_q$ 上的椭圆曲线 E 上的一个点。这一计算称为点乘或标量乘，它决定着椭圆曲线密码体制的运算速度（见第 4 章）。这里所讨论的技术不依赖于任何具体的曲线结构。3.4 节、3.5 节和 3.6 节中将讨论对于一些特殊曲线具有有效可计算自同态的点乘的计算方法。3.3.1 节讨论 P 事先不知道时的情况。在一些实例中 P 是固定的，如在 ECDSA 签名生成中（见 4.4.1 节），点乘算法可以利用预计算数据，这些数据仅仅取决于 P，而不取决于 k，这种算法将在 3.3.2 节给出。计算 $kP + lQ$ 的有效算法将在 3.3.3 节给出。这种计算称为多点乘，它决定着某些椭圆曲线密码体制的效率，例如 ECDSA 的签名验证（见 4.4.1 节）。

我们假设 $\#E(\mathbb{F}_q) = nh$，其中 n 是素数，h 是一个小整数（因此 $n \approx q$）。P 和 Q 的阶

为 n，乘数 k 随机地从区间 $[1, n-1]$ 内选择。k 的二进制表示为 $(k_{t-1}, \ldots, k_2, k_1, k_0)_2$，其中 $t \approx m = \lceil \mathrm{lb}\, q \rceil$。

3.3.1　未知点

算法 3.26 和算法 3.27 是计算幂的反复平方乘算法的扩充。算法 3.26 从右到左处理 k 的各位，而算法 3.27 却是从左向右处理。

算法 3.26　计算点乘的从右向左的二进制方法

输入：$k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$。

输出：kP。

1. $Q \leftarrow \infty$。
2. 对于 i 从 0 到 $t-1$，重复执行
 2.1 若 $k_i = 1$，则 $Q \leftarrow Q + P$。
 2.2 $P \leftarrow 2P$。
3. 返回(Q)。

算法 3.27　计算点乘的从左向右的二进制方法

输入：$k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$。

输出：kP。

1. $Q \leftarrow \infty$。
2. 对于 i 从 $t-1$ 到 0，重复执行
 2.1 $Q \leftarrow 2Q$。
 2.2 若 $k_i = 1$，则 $Q \leftarrow Q + P$。
3. 返回(Q)。

k 的二进制表示中 1 的个数的期望值为 $t/2 \approx m/2$，因而算法 3.27 的期望运行时间近似为 $m/2$ 次点加和 m 次倍点，表示为

$$\frac{m}{2}A + mD。 \tag{3.15}$$

令 M 表示域乘法，S 表示域平方，I 表示域的逆。若域 $\mathbb{F}_q$ 的特征大于 3 且采用仿射坐标（见 3.1.2 节），则用域运算来表示的运行时间为

$$2.5mS+3mM+1.5mI; \tag{3.16}$$

而对于二进制域，则运行时间又可表示为

$$3mM+1.5mI。 \tag{3.17}$$

假设域$\mathbb{F}_q$的特征大于3。

若采用混合坐标（见3.2.2节），则Q以雅可比坐标存储，而P以仿射坐标存储。因此，算法中步骤2.1的倍点计算可以应用算法3.21来实现，而步骤2.2中的点加计算可以应用算法3.22来实现。算法3.27的域运算的量为

$$8mM+5.5mS+(1I+3M+1S) \tag{3.18}$$

（还原成仿射坐标需要1次求逆、3次乘法和1次平方）。

假设$\mathbb{F}_q$是二进制域。若采用混合坐标（见3.2.3节），则Q以LD投影坐标存储，而P以仿射坐标存储。因此，算法中步骤2.1的倍点计算可以应用算法3.24来实现，而步骤2.2中的点加计算可以应用算法3.25来实现。算法3.27的域运算的量为

$$8.5mM+(2M+1I) \tag{3.19}$$

（还原成仿射坐标需要1次求逆和2次乘法）。

非相邻表示型（NAF）

设$P=(x,y)\in E(\mathbb{F}_q)$。若$\mathbb{F}_q$是二进制域，则$-P=(x,x+y)$；若$\mathbb{F}_q$的特征大于3，则$-P=(x,-y)$。因此椭圆曲线点的减法与点的加法一样有效。这促使我们使用k的带符号数字表示$k=\sum_{i=0}^{l-1}k_i2^i$，其中$k_i\in\{0,\pm1\}$。一种特别有用的带符号数字表示是非相邻表示型(NAF)。

定义 3.28 一个正整数k的非相邻表示型是表达式$k=\sum_{i=0}^{l-1}k_i2^i$，其中$k_i\in\{0,\pm1\}$，$k_{l-1}\neq0$，并且没有两个连续的数字k_i是非零的。NAF的长度是l。

定理 3.29（NAF的性质）令k是一个正整数。

(i) k有惟一的NAF，并记为NAF(k)。

(ii) NAF(k)在k的所有带符号表示中具有最少的非零位。

(iii) NAF(k)的长度最多比k的二进制表示的长度大1。

(iv) 若NAF(k)的长度是l，则$2^l/3<k<2^{l+1}/3$。

(v) 所有长度为l的NAF中非零数字的平均密度约为1/3。

利用算法 3.30 可以有效地计算 NAF(k)。NAF(k)的各位数字可通过连续用 2 去除 k 且允许余数取 0 或±1 来得到。若 k 是奇数，则余数 $r \in \{-1, 1\}$，以使商 $(k-r)/2$ 是偶数，这将确保下一个 NAF 数字是 0。

算法 3.30　计算一个正整数的 NAF

输入：一个正整数 k。

输出：NAF(k)。

1. $i \leftarrow 0$。
2. 当 $k \geqslant 1$ 时，重复执行
 2.1 若 k 是奇数，则 $k_i \leftarrow 2-(k \bmod 4), k \leftarrow k-k_i$。
 2.2 否则，$k_i \leftarrow 0$。
 2.3 $k \leftarrow k/2, i \leftarrow i+1$。
3. 返回$(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$。

算法 3.31 修改了从左到右的二进制点乘方法（算法 3.27），用 NAF(k)代替了 k 的二进制表示。这导致了定理 3.29 的 (iii) 和 (iv) 成立，算法 3.31 的期望运行时间近似为

$$\frac{m}{3}A + mD。 \tag{3.20}$$

算法 3.31　用二进制 NAF 方法计算点乘

输入：一个正整数 k，$P \in E(\mathbb{F}_q)$。

输出：kP。

1. 用算法 3.30 计算$\mathrm{NAF}(k) = \sum_{i=0}^{l-1} k_i 2^i$。
2. $Q \leftarrow \infty$。
3. 对于 i 从 $l-1$ 到 0，重复执行
 3.1 $Q \leftarrow 2Q$。
 3.2 若 $k_i = 1$，则 $Q \leftarrow Q+P$。
 3.3 若 $k_i = -1$，则 $Q \leftarrow Q-P$。
4. 返回(Q)。

窗口方法

若可以利用额外的存储器，采用一次处理 k 的 w 位的窗口方法，则算法 3.31 的运行时间将可以减少。

定义 3.32 令 $w \geqslant 2$ 是正整数。正整数 k 的宽度为 w 的 NAF 是表达式 $k=\sum_{i=0}^{l-1} k_i 2^i$，其中每一个非零系数 k_i 都是奇数，$|k_i|<2^{w-1}$，$k_{l-1}\neq 0$，并且任何连续 w 个数字中最多 1 位为非零。宽度为 w 的 NAF 的长度是 l。

定理 3.33 （宽度 w 的 NAF 的性质）令 k 是一个正整数。

(i) k 有惟一的宽度 w 的 NAF，并记为 $\mathrm{NAF}_w(k)$。

(ii) $\mathrm{NAF}_2(k)=\mathrm{NAF}(k)$。

(iii) $\mathrm{NAF}_w(k)$的长度最多比 k 的二进制表示的长度大 1。

(iv) 在所有长度为 l 的宽度 w 的 NAF 中，平均非零数字的密度近似为 $1/(w+1)$。

例 3.34 （宽度 w 的 NAF）令 $k=1\,122\,334\,455$。我们用 $\bar{c}$ 表示负整数 $-c$。k 的二进制表示及 k 的宽度 $w(2\leqslant w\leqslant 6)$的 NAF 如下：

$$
\begin{aligned}
(k)_2 &= 1\,0\,0\,0\;0\,1\,0\,1\,1\;1\,0\,0\,1\,0\,1\;0\,1\,1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,1\,1\,1\\
\mathrm{NAF}_2(k) &= 1\,0\,0\,0\;1\,0\,\bar{1}\,0\,0\;\bar{1}\,0\,1\,0\,\bar{1}\,0\;\bar{1}\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\,0\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\\
\mathrm{NAF}_3(k) &= 1\,0\,0\,0\;0\,0\,3\,0\,0\;\bar{1}\,0\,0\,1\,0\,0\;3\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\,0\,0\,0\,0\,\bar{1}\,0\,0\,\bar{1}\\
\mathrm{NAF}_4(k) &= 1\,0\,0\,0\;0\,1\,0\,0\,0\;7\,0\,0\,0\,0\,5\;0\,0\,0\,7\,0\,0\,0\,7\,0\,0\,0\,\bar{1}\,0\,0\,0\,7\\
\mathrm{NAF}_5(k) &= 1\,0\,0\,0\,0\,\overline{15}\,0\,0\,0\,0\;\bar{9}\,0\,0\,0\,0\,0\,11\,0\,0\,0\,0\,0\,0\,\bar{9}\,0\,0\,0\,0\,0\,0\,0\,\bar{9}\\
\mathrm{NAF}_6(k) &= 1\,0\,0\,0\;0\,0\,0\,0\,0\,23\,0\,0\,0\,0\,0\,11\,0\,0\,0\,0\,0\,0\,\bar{9}\,0\,0\,0\,0\,0\,0\,0\,\bar{9}
\end{aligned}
$$

利用算法 3.35 可以有效地计算 $\mathrm{NAF}_w(k)$，这里 $k \bmod 2^w$ 表示一个正整数 u，u 满足 $u\equiv k \pmod{2^w}$且 $-2^{w-1}\leqslant u<2^{w-1}$。$\mathrm{NAF}_w(k)$的各位数字通过 k 连续除 2 并使余数 r 在区间$[-2^{w-1},\ 2^{w-1}-1]$内来获得。若 k 是奇数并且余数 $r=k \bmod 2^w$，则$(k-r)/2$将能被 2^{w-1} 整除，从而使后面的 $w-1$ 个数字为零。

算法 3.35 计算一个正整数的窗口宽度 w 的 NAF

输入：窗口宽度 w，一个正整数 k。

输出：$\mathrm{NAF}_w(k)$。

1. $i\leftarrow 0$。
2. 当 $k\geqslant 1$ 时，重复执行

 2.1 若 k 是奇数，则 $k_i\leftarrow k \bmod 2^w, k\leftarrow k-k_i$。

 2.2 否则，$k_i\leftarrow 0$。

 2.2 $k\leftarrow k/2, i\leftarrow i+1$。
3. 返回$(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$。

算法 3.36 通过用 $\mathrm{NAF}_w(k)$ 代替 $\mathrm{NAF}(k)$，扩展了二进制 NAF 方法（算法 3.31）。若依据定理 3.33 的 (iii) 和 (iv)，则算法 3.36 的期望运行时间近似为

$$\left[1D+(2^{w-2}-1)A\right]+\left[\frac{m}{w+1}A+mD\right]。\tag{3.21}$$

算法 3.36　计算点乘的窗口 NAF 方法

输入：窗口宽度 w，一个正整数 k，$P\in E(\mathbb{F}_q)$。

输出：kP。

1. 用算法 3.35 计算 $\mathrm{NAF}_w(k)=\sum_{i=0}^{l-1}k_i2^i$。
2. 对于 $i\in\{1,3,5,\ldots,2^{w-1}-1\}$，计算 $P_i=iP$。
3. $Q\leftarrow\infty$。
4. 对于 i 从 $l-1$ 到 0，重复执行
 4.1 $Q\leftarrow 2Q$。
 4.2 若 $k_i\neq 0$，则
 　　若 $k_i>0$，则 $Q\leftarrow Q+P_{k_i}$。
 　　否则，$Q\leftarrow Q-P_{-k_i}$。
5. 返回(Q)。

注 3.37　（坐标选择）通过为累加数 Q 采用投影坐标，可使所需的求逆的次数下降。若域上的求逆比域上的乘法费时，则投影坐标还对计算 P_i 有效。对素域上的椭圆曲线采用 Chudnovsky 坐标（见 3.2.2 节）可以省去预计算中的求逆，这样做的代价是在赋值阶段雅可比- Chudnovsky 混合坐标的加法的效率较低。

窗口 NAF 采用“滑动窗口”技术，算法 3.35 采用了宽度为 w 的窗口，并且从右向左进行，跳过连续的零位，只对非零位 k_i 进行处理。作为一种选择，将滑动窗口技术用于 k 的 NAF，导致产生了算法 3.38。窗口（宽度最多为 w）从左向右沿 $\mathrm{NAF}(k)$ 的数字滑动，窗口的位置要使窗口内的值为奇数（省掉所需的预计算）。

算法 3.38　计算点乘的滑动窗口方法

输入：窗口宽度 w，正整数 k，$P\in E(\mathbb{F}_q)$。

输出：kP。

1. 用算法 3.30 计算 $\mathrm{NAF}(k)=\sum_{i=0}^{l-1}k_i2^i$。
2. 对于 $i\in\{1,3,\ldots,2(2^w-(-1)^w)/3-1\}$，计算 $P_i=iP$。

3. $Q \leftarrow \infty, i \leftarrow l-1$。

4. 当 $i \geqslant 0$ 时，重复执行

4.1 若 $k_i = 0$，则 $t \leftarrow 1, u \leftarrow 0$。

4.2 否则，寻找一个最大的 $t \leqslant w$ 使得 $u \leftarrow (k_i, \ldots, k_{i-t+1})$ 是奇数。

4.3 $Q \leftarrow 2^t Q$。

4.4 若 $u>0$，则 $Q \leftarrow Q + P_u$；否则，若 $u<0$，则 $Q \leftarrow Q - P_{-u}$。

4.5 $i \leftarrow i-t$。

5. 返回(Q)。

在滑动窗口方法中零在窗口之间移动的平均长度是

$$v(w) = \frac{4}{3} - \frac{(-1)^w}{3 \cdot 2^{w-2}}。$$

由此得出，算法 3.38 的期望运行时间近似为

$$\left[1D + \left(\frac{2^w - (-1)^w}{3} - 1\right) A\right] + \frac{m}{w + v(w)} A + mD。 \qquad (3.22)$$

注 3.39 （*滑动窗口方法与窗口 NAF 方法比较*）对于给定的 w，滑动窗口方法与宽度 w 的 NAF 方法相比，滑动窗口方法允许窗口内的数更大。这说明滑动窗口方法预计算的开销较高（粗略计算表明，在算法 3.38 的步骤 2 中有 $2^w/3$ 次点乘，而在算法 3.36 的步骤 2 中有 $2^w/4$ 次点乘。），但是在主循环中滑动窗口方法的点乘次数比宽度 w 的 NAF 方法要少，分别约有 $(m/(w+v(w)))$ 次和 $(m/(w+1))$ 次。若比较点的运算，则对于感兴趣的 m 值窗口 NAF 方法使用的点加较少（每种方法都选用最佳的 w 值）。要进行精确比较，还要考虑采用不同坐标系的求逆和乘法的比较。

作为一个例子，考虑二进制 NIST 曲线，并假设求逆与乘法的开销比为 $I/M = 8$。预计算采用仿射坐标，而主循环采用投影和仿射混合坐标进行点加。表 3.5 给出了每一种方法点加的期望开销。注意，对于每一种方法都还有 m 次倍点。因此各种方法点乘的时间差别略小于表 3.5 中所给出的值。若要限制预计算时所要存储的点数，则预计算的差别便可确定出哪种方法最好。例如，只允许存储 3 个点，则滑动窗口方法就是最好的，而若允许存储 4 个点，则窗口 NAF 方法将好一些。然而差别相当小；在本例中，两种方法都用 $w=3$（分别有 2 个或 3 个点的预计算），但滑动窗口方法较好，它比窗口 NAF 方法节省了 7%~10% 的点乘开销。

表 3.5　当 $l/M=8$ 时，滑动窗口与窗口 NAF 方法在点加计算中的开销。其中“点数”指在预计算时要存储的点数，“WN”表示窗口 NAF 方法（算法 3.36），“SW”表示滑动窗口方法（算法 3.38）

w	点数		$m=163$		$m=233$		$m=283$		$m=409$		$m=571$	
	WN	SW	WN	SW	WN	SW	WN	SW	WN	SW	WN	SW
2	1	1	442	442	626	626	762	762	1098	1098	1530	1530
3	2	3	340	318	484	438	580	526	836	750	1156	1038
4	4	5	296	298	408	402	488	474	688	666	952	914
5	8	11	296	310	384	398	456	462	624	622	840	822
6	16	21	344	386	424	458	480	514	624	650	808	834

Montgomery 方法

二进制域上非超奇异椭圆曲线 $y^2+xy=x^3+ax^2+b$ 的算法 3.40 是由 López 和 Dahab 提出的。其基本思想基于 Montgomery 方法。令 $Q_1=(x_1,y_1)$ 和 $Q_2=(x_2,y_2)$，$Q_1\neq\pm Q_2$，再令 $Q_1+Q_2=(x_3,y_3)$，$Q_1-Q_2=(x_4,y_4)$，则利用加法公式（见 3.1.2 节）可以验证

$$x_3=x_4+\frac{x_2}{x_1+x_2}+\left(\frac{x_2}{x_1+x_2}\right)^2。\tag{3.23}$$

因此，Q_1+Q_2 的 x 坐标可以由 Q_1，Q_2 和 Q_1-Q_2 的 x 坐标来计算。确定 kP 的算法 3.40 中的 j 迭代仅计算 $T_j=[lP,(l+1)P]$ 的 x 坐标，其中 l 是 k 的最左 j 位所表示的整数。若 k 的最左 $(j+1)$ 位为 0 或 1，则 $T_{j+1}=[2lP,(2l+1)P]$ 或 $[(2l+1)P,(2l+2)P]$，如图 3.4 所示。每次迭代都需要使用式 (3.23) 的 1 次倍点和 1 次点加。最后一次迭代后，计算出 $kP=(x_1,y_1)$ 和 $(k+1)P=(x_2,y_2)$ 的 x 坐标，而 kP 的 y 坐标可如下恢复：

$$y_1=x^{-1}(x_1+x)[(x_1+x)(x_2+x)+x^2+y]+y\tag{3.24}$$

式 (3.24) 是利用从 $kP=(x_1,y_1)$ 和 $P=(x,y)$ 计算 $(k+1)P$ 的 x 坐标 x_2 的公式导出的。

$$\underbrace{(k_{t-1}k_{t-2}\cdots k_{t-j}}_{\downarrow}\ \underbrace{k_{t-(j+1)}}_{\downarrow}\ k_{t-(j+2)}\cdots k_1k_0)_2P$$

$$[lP,(l+1)P]\rightarrow\boxed{\begin{array}{ll}[2lP,lP+(l+1)P], & \text{若 } k_{t-(j+1)}=0\\ [lP+(l+1)P,2(l+1)P], & \text{若 } k_{t-(j+1)}=1\end{array}}$$

图 3.4　Montgomery 方法点乘的一次迭代。j 次迭代后，对于 $l=(k_{t-1}\cdots k_{t-j})_2$，$lP$ 和 $(l+1)P$ 的 x 坐标便知道了。$j+1$ 次迭代需要 1 次倍点和 1 次点加运算，以便为 $l'=(k_{t-1}\cdots k_{t-(j+1)})_2$ 寻找 $l'P$ 和 $(l'+1)P$ 的 x 坐标

算法 3.40 是利用标准投影坐标（见 3.2.1 节）给出的；在算法的步骤 1 和步骤 2 仅计算点的 X 坐标和 Z 坐标。期望运行时间近似为

$$6mM+(1I+10M)。 \tag{3.25}$$

算法 3.40 的一个优点是不需要额外的存储器。另一个优点是在主循环的每一次迭代中都完成相同的操作，因此增强了抵抗时间分析攻击和能量分析攻击的能力（见 5.3 节）。

算法 3.40 Montgomery 点乘（对于域 $\mathbb{F}_{2^m}$ 上的椭圆曲线）

输入：$k=(k_{t-1},\ldots,k_1,k_0)_2$，其中 $k_{t-1}=1$，$P=(x,y)\in E(\mathbb{F}_{2^m})$，$k<n$。

输出：kP。

1. $X_1\leftarrow x$，$Z_1\leftarrow 1$，$X_2\leftarrow x^4+b$，$Z_2\leftarrow x^2$。{计算 $(P,2P)$}
2. 对于 i 从 $t-2$ 到 0，重复执行
 2.1 若 $k_i=1$，则
 $T\leftarrow Z_1$，$Z_1\leftarrow (X_1Z_2+X_2Z_1)^2$，$X_1\leftarrow xZ_1+X_1X_2TZ_2$。
 $T\leftarrow X_2$，$X_2\leftarrow X_2^4+bZ_2^4$，$Z_2\leftarrow T^2Z_2^2$。
 2.2 否则
 $T\leftarrow Z_2$，$Z_2\leftarrow (X_1Z_2+X_2Z_1)^2$，$X_2\leftarrow xZ_2+X_1X_2Z_1T$。
 $T\leftarrow X_1$，$X_1\leftarrow X_1^4+bZ_1^4$，$Z_1\leftarrow T^2Z_1^2$。
3. $x_3\leftarrow X_1/Z_1$。
4. $y_3\leftarrow (x+X_1/Z_1)[(X_1+xZ_1)(X_2+xZ_2)+(x^2+y)(Z_1Z_2)](xZ_1Z_2)^{-1}+y$。
5. 返回 (x_3,y_3)。

3.3.2 固定点

若点 P 是固定的，并且有存储器可用，则通过对某些数据的预计算，点乘 kP 的计算速度将会提高。例如，若预计算出 $2P,2^2P,\ldots,2^{t-1}P$，则从右向左的二进制方法（算法 3.26）的运算时间可望提高到 $(m/2)A$（省去了所有的倍点运算）[①]。

固定基窗口方法

Brickell, Gordon, McCurley 和 Wilson 提出了如下预计算 2^iP 的简单且优雅的方法。令

① 回忆下列标注：t 是 k 的位长，$m=\lceil \mathrm{lb}\, q\rceil$，并假设 $t\approx m$。

$(K_{d-1},\ldots,K_1,K_0)_{2^w}$ 是 k 的以 2^w 为基的表达式，其中 $d=\lceil t/w\rceil$。再令 $Q_j=\sum_{i:K_i=j}2^{wi}P$，对于每个 $1\leqslant j\leqslant 2^w-1$，则

$$kP=\sum_{i=0}^{d-1}K_i(2^{wi}P)=\sum_{j=1}^{2^w-1}\left(j\sum_{i:K_i=j}2^{wi}P\right)=\sum_{j=1}^{2^w-1}jQ_j$$
$$=Q_{2^w-1}+(Q_{2^w-1}+Q_{2^w-2})+\cdots+(Q_{2^w-1}+Q_{2^w-2}+\cdots+Q_1)。$$

算法 3.41 就是基于这一推证。期望运行时间近似为

$$(2^w+d-3)A, \tag{3.26}$$

其中 $d=\lceil t/w\rceil$ 且 $t\approx m$。

算法 3.41　计算点乘的固定基窗口方法

输入：窗口宽度 $w, d=\lceil t/w\rceil, k=(K_{d-1},\ldots,K_1,K_0)_{2^w}, P\in E(\mathbb{F}_q)$。

输出：kP。

1. 预计算 $P_i=2^{wi}P$，$0\leqslant i\leqslant d-1$。
2. $A\leftarrow\infty, B\leftarrow\infty$。
3. 对于 j 从 2^w-1 到 1，重复执行
 3.1 对于每个使 $K_i=j$ 的 i，执行：$B\leftarrow B+P_i$。{Q_j 与 B 相加}
 3.2 $A\leftarrow A+B$。
4. 返回(A)。

算法 3.42 通过用 NAF(k) 代替了 k 的二进制表达式，改进了算法 3.41。在算法 3.42 中，NAF(k) 被划分为符号属于 $\{0,\pm1\}$ 的符号串 K_i，每个串都有相同的长度 w：

$$\text{NAF}(k)=K_{d-1}\|\cdots\|K_1\|K_0。$$

因为每个 K_i 都处于非相邻表示形中，所以它可表示成区间 $[-I,I]$ 内的一个整数；其中，若 w 是偶数，则 $I=(2^{w+1}-1)/3$，而若 w 是奇数，则 $I=(2^{w+1}-1)/3$。算法 3.42 的期望运行时间近似为

$$\left(\frac{2^{w+1}}{3}+d-2\right)A, \tag{3.27}$$

其中 $d=\lceil (t+1)/w\rceil$。

算法 3.42 计算点乘的固定基 NAF 窗口方法

输入：窗口宽度 w，正整数 k，$P \in E(\mathbb{F}_q)$。

输出：kP。

1. 预计算。计算 $P_i = 2^{wi}P$，$0 \leqslant i \leqslant \lceil (t+1)/w \rceil$。
2. 用算法 3.30 计算 $\mathrm{NAF}(k) = \sum_{i=0}^{l-1} k_i 2^i$。
3. $d \leftarrow \lceil l/w \rceil$。
4. 若需要，则用 0 填充 NAF(k) 的左端。记 $(k_{l-1}, \ldots, k_1, k_0) = K_{d-1} \| \cdots \| K_1 \| K_0$，其中每个 K_i 都是长度为 d 的 $\{0, \pm 1\}$ 符号串。
5. 若 w 是偶数，则 $I \leftarrow (2^{w+1}-2)/3$；否则，$I \leftarrow (2^{w+1}-1)/3$。
6. $A \leftarrow \infty, B \leftarrow \infty$。
7. 对于 j 从 I 到 1，重复执行
 7.1 对于每个使 $K_i = j$ 的 i，执行：$B \leftarrow B + P_i$。 {Q_j 与 B 相加}
 7.2 对于每个使 $K_i = -j$ 的 i，执行：$B \leftarrow B - P_i$。 {$-Q_j$ 与 B 相加}
 7.3 $A \leftarrow A + B$。
8. 返回(A)。

注 3.43（坐标的选择）若域上的求逆特别费时，则算法 3.41 和算法 3.42 中的累加器 A 或 B 采用投影坐标是非常有效的。对于素域上的椭圆曲线，表 3.3 表明，若采用投影坐标，则 B 采用 Chudnovsky 坐标而 A 采用雅可比坐标是非常好的。在这种情况下，算法 3.42 中的步骤 7.1 和步骤 7.2 使用 Chudnovsky-仿射混合坐标的加法，而在步骤 7.3 使用雅可比- Chudnovsky 混合坐标的加法。

固定基的 comb 方法

令 $d = \lceil t/w \rceil$。在混合基 Comb 方法（算法 3.44）中，首先在 k 的二进制表示的左边填充 $dw - t$ 个 0，然后划分成个 w 个二进制串，每个串有 d 位，并且使得

$$k = K^{w-1} \| \cdots \| K^1 \| K^0。$$

将二进制串 K^j 写为代表矩阵的行

$$\begin{bmatrix} K^0 \\ \vdots \\ K^{w'} \\ \vdots \\ K^{w-1} \end{bmatrix} = \begin{bmatrix} K_{d-1}^0 & \cdots & K_0^0 \\ \vdots & & \vdots \\ K_{d-1}^{w'} & \cdots & K_0^{w'} \\ \vdots & & \vdots \\ K_{d-1}^{w-1} & \cdots & K_0^{w-1} \end{bmatrix} = \begin{bmatrix} k_{d-1} & \cdots & k_0 \\ \vdots & & \vdots \\ k_{(w'+1)d-1} & \cdots & k_{w'd} \\ \vdots & & \vdots \\ k_{wd-1} & \cdots & k_{(w-1)d} \end{bmatrix},$$

其中各列同时处理。为了提高计算速度，对于所有可能的位串 $(a_{w-1},\ldots,a_1,a_0)$，对点

$$[a_{w-1},\ldots,a_2,a_1,a_0]P = a_{w-1}2^{(w-1)d}P+\cdots+a_2 2^{2d}P+a_1 2^d P+a_0 P$$

进行预计算。

算法 3.44　计算点乘的固定基 comb 方法

输入：窗口宽度 $w, d=\lceil t/w\rceil$，$k=(k_{t-1},\ldots,k_1,k_0)_2$，$P\in E(\mathbb{F}_q)$。

输出：kP。

1. 对于所有长度为 w 的二进制串 $(a_{w-1},\ldots,a_1,a_0)$，预计算 $[a_{w-1},\ldots,a_1,a_0]P$。
2. 若需要，则用 0 填充 k 的左边，记 $k=K^{w-1}\|\cdots\|K^1\|K^0$，其中每个串 K^j 的长度为 d。用 K_i^j 表示 K^j 的第 i 位。
3. $Q\leftarrow\infty$。
4. 对于 i 从 $d-1$ 到 0，重复执行
 4.1 $Q\leftarrow 2Q$。
 4.2 $Q\leftarrow Q+[K_i^{w-1},\ldots,K_i^1,K_i^0]P$。
5. 返回(Q)。

算法 3.44 的期望运行时间为

$$\left(\frac{2^w-1}{2^w}d-1\right)A+(d-1)D。\qquad(3.28)$$

对于 $w>2$，算法 3.44 的主循环中有近似相等的点加和倍点运算量。图 3.5 示出了算法 3.45 中使用了第二个预计算表，从而使倍点的运算量较之点加运算量减半。

算法 3.45　计算点乘的固定基 comb 方法（使用两个预计算表）

输入：窗口宽度 $w, d=\lceil t/w\rceil, e=\lceil d/2\rceil, k=(k_{t-1},\ldots,k_0)_2, P\in E(\mathbb{F}_q)$。

输出：kP。

1. 对于所有长度为 w 的二进制串 $(a_{w-1},\ldots,a_1,a_0)$，预计算 $[a_{w-1},\ldots,a_1,a_0]P$ 和 $2^e[a_{w-1},\ldots,a_1,a_0]P$。
2. 若需要，则用 0 填充 k 的左边，记 $k=K^{w-1}\|\cdots\|K^1\|K^0$，其中每个 K^j 是长度为 d 的位串。用 K_i^j 表示 K^j 的第 i 位。
3. $Q\leftarrow\infty$。
4. 对于 i 从 $e-1$ 到 0，重复执行

4.1 $Q \leftarrow 2Q$。

4.2 $Q \leftarrow Q + [K_i^{w-1}, \ldots, K_i^1, K_i^0]P + 2^e[K_{i+e}^{w-1}, \ldots, K_{i+e}^1, K_{i+e}^0]P$。

5. 返回(Q)。

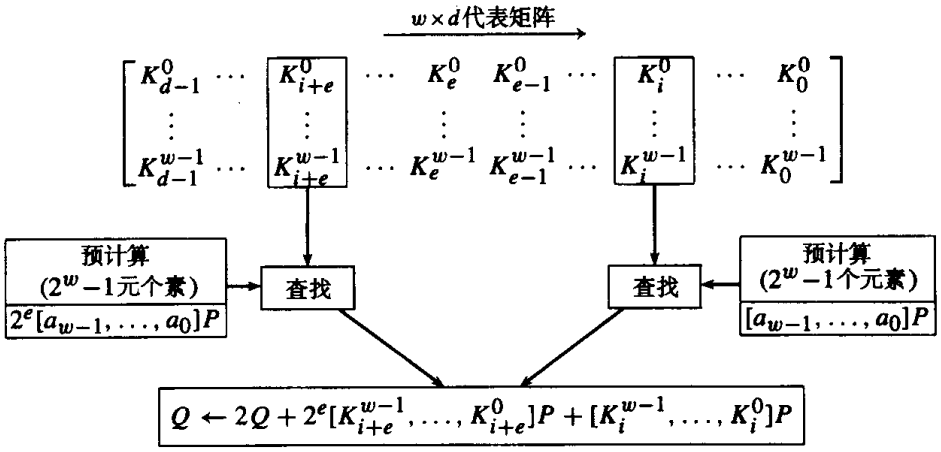


图 3.5 算法 3.45 的一次迭代。$w \times d$ 代表矩阵从左向右处理，实施 $e = \lceil d/2 \rceil$ 次迭代以求得 kP。对于所有长度为 w 的二进制串 $(a_{w-1}, \ldots, a_0)$，预计算 $[a_{w-1}, \ldots, a_0]P$ 和 $2^e[a_{w-1}, \ldots, a_0]P$，其中 $[a_{w-1}, \ldots, a_0] = a_{w-1}2^{(w-1)d} + \cdots + a_1 2^d + a_0$

算法 3.45 的期望运行时间为

$$\left(\frac{2^w - 1}{2^w}2e - 1\right)A + (e-1)D。 \tag{3.29}$$

对于固定的 w，算法 3.45 的预计算需要的存储器两倍于算法 3.44。对于给定量的预计算，算法 3.45 将优于算法 3.44，条件是

$$\frac{2^{w-1}(w-2)}{2^w - w - 1} \geqslant \frac{A}{D},$$

其中 w 是算法 3.44 使用的窗口宽度（算法 3.45 使用的宽度是 $w-1$）。作为一个例子，在二进制域情况下的 LD 坐标给出 $A/D \approx 2$，为了使两个表的方法更优，在算法 3.44 中近似需要 $w \geqslant 6$。对于素域上的 NIST 曲线，$A/D \approx 1.4$，对于雅可比坐标和 $S = 0.8M$，则需要 $w \geqslant 4$。

注 3.46 （具有同时加的算法 3.45）若容许存储 e 个点（取决于 k），则算法 3.45 中的步骤 4.2 的值

$$T_i \leftarrow [K_i^{w-1}, \ldots, K_i^1, K_i^0]P + 2^e[K_{i+e}^{w-1}, \ldots, K_{i+e}^1, K_{i+e}^0]P, \quad 0 \leqslant i < e$$

就能在预计算中确定。点 T_i 按仿射坐标计算，采用算法 2.26 的同时求逆方法，只需要计算 1 次求逆和 $3(e'-1)$ 次域的乘法，而不是 $e'=(1-1/2^w)^2e$ 次求逆。

若 Q 是投影坐标，则在步骤 4.2 中的 e' 次混合坐标加将被预计算阶段的 e' 次同时加所代替。对于 3.2 节讨论的坐标，这将得到下面的近似域运算量。

e'次$E(\mathbb{F}_{2^m})$上的加		e'次$E(\mathbb{F}_{p^m})$上的加，$p>3$	
混合坐标	同时	混合坐标	同时
$8e'M$	$I+(5e'-3)M$	$8e'M+3e'S$	$I+(5e'-3)M+e'S$

对于 3.2 节讨论的二进制域和 OEF 上实际感兴趣的曲线，I/M 是很小的，预期可节约 15% 的点乘时间。

注 3.47　（comb 方法）算法 3.44 和算法 3.45 是由 Lim 和 Lee 提出的指数运算方法的特例。对于给定的参数 w 和 v，一个 t 位的整数 k 被写为一个 $w\times d$ 代表矩阵的行，其中 $d=\lceil t/w\rceil$，如图 3.6 所示。一个典型的矩阵元素 $K_{v'}^{w'}$ 由 k 的 $e=\lceil d/v\rceil$ 位组成，$K_{v'}^{w'}=(k_{l+e-1},\ldots,k_{l+1},k_l)$，其中 $l=dw'+ev'$（若 $v'=v-1$ 且 $v\nmid d$，则用零代替矩阵元素）。

$$
w\left\{\begin{array}{cccccc}
K_{v-1}^{0} & \cdots & (K_{v',e-1}^{0},\ldots,K_{v',0}^{0}) & \cdots & K_{0}^{0}\\
\vdots & & \vdots & & \vdots\\
K_{v-1}^{w'} & \cdots & (K_{v',e-1}^{w'},\ldots,K_{v',0}^{w'}) & \cdots & K_{0}^{w'}\\
\vdots & & \vdots & & \vdots\\
K_{v-1}^{w-1} & \cdots & (\underbrace{K_{v',e-1}^{w-1},\ldots,K_{v',0}^{w-1}}_{e=\lceil d/v\rceil\text{位}}) & \cdots & K_{0}^{w-1}
\end{array}\right.
\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{d=\lceil t/w\rceil\text{位}}
$$

图 3.6　Lim-Lee 的 comb 方法的代表矩阵。对于给定的参数 w 和 v，一个 t 位的整数 k 被写为一个 $w\times d$ 代表矩阵的行，其中 $d=\lceil t/w\rceil$。矩阵元素 $K_{v'}^{w'}$ 由 k 的 $e=\lceil d/v\rceil$ 位组成

若矩阵元素 $K^{w'}$ 表示 w' 行组成的整数，则

$$
\begin{aligned}
kP&=\sum_{w'=0}^{w-1}K^{w'}2^{dw'}P=\sum_{w'=0}^{w-1}\Big(\sum_{v'=0}^{v-1}K_{v'}^{w'}2^{ev'}\Big)2^{dw'}P\\
&=\sum_{w'=0}^{w-1}\sum_{v'=0}^{v-1}\Big(\sum_{e'=0}^{e-1}K_{v',e'}^{w'}2^{e'}\Big)2^{ev'}2^{dw'}P\\
&=\sum_{e'=0}^{e-1}2^{e'}\sum_{v'=0}^{v-1}\Big(\underbrace{2^{ev'}\sum_{w'=0}^{w-1}K_{v',e'}^{w'}2^{dw'}P}_{P[v'][K_{v',e'}]}\Big),
\end{aligned}
$$

其中$v(2^w-1)$个点$P[v'][u]$，$v'\in[0,v-1]$，$u\in[1,2^w-1]$要进行预计算。基于这种方法的点乘算法近似需要$e-1\approx\frac{t}{wv}-1$次倍点和$(\frac{t}{w}-1)\frac{2^w-1}{2^w}$次点加。算法 3.44 和算法 3.45 分别是$v=1$和$v=2$的情况。

3.3.3 多点乘

一种提高计算（$kP+lQ$）速度的方法是同时多点乘的算法（算法 3.48），此外它还被称为 Shamir 窍门方法。若 k 和 l 是 t 位数，则它的二进制表示可写成一个$2\times t$代表矩阵。给定宽度w，对于$0\leqslant i,j<2^w$，计算值$iP+jQ$。在$\lceil t/w\rceil$步中的每一步，累加器从值$iP+jQ$的表接收w次倍点和 1 次点加，而这些值由移过代表矩阵的$2\times w$窗口来确定；如图 3.7 所示。

算法 3.48 的期望运行时间近似为

$$\left[(3\cdot 2^{2(w-1)}-2^{w-1}-1)A+(2^{2(w-1)}-2^{w-1})D\right]+\left[\left(\frac{2^{2w}-1}{2^{2w}}d-1\right)A+(d-1)wD\right],\quad(3.30)$$

并且需要存储$2^{2w}-1$个点。

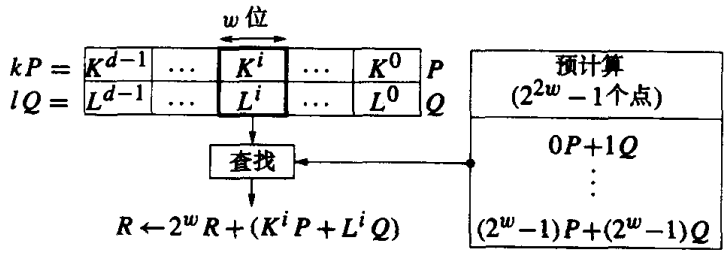


图 3.7 同时点乘累加步

算法 3.48 同时多点乘

输入：窗口宽度w，$k=(k_{t-1},\ldots,k_0)_2$，$l=(l_{t-1},\ldots,l_0)_2$，$P,Q\in E(\mathbb{F}_q)$。

输出：$kP+lQ$。

1. 对于所有$i,j\in[0,2^w-1]$，计算$iP+jQ$。
2. 记$k=(K^{d-1},\ldots,K^1,K^0)$，$l=(L^{d-1},\ldots,L^1,L^0)$，其中每个$K^i$，$L^i$是长度为$w$的位串，$d=\lceil t/w\rceil$。
3. $R\leftarrow\infty$。
4. 对于i从$d-1$到 0，重复执行

 4.1 $R\leftarrow 2^wR$。

4.2　$R \leftarrow R + (K^i P + L^i Q)$。

5．返回(R)。

算法 3.48 通过使用滑动窗口可以得到改善。每一步放置一个宽度最多为w的窗口，使得最右边的列为非零。预计算所需要的存储量减少了$2^{2(w-1)}-1$个点。改进后的算法在赋值阶段期望有$t/(w+(1/3))$次点加，与$w \in \{2,3\}$的算法 3.48 相比近似节省 9%（在赋值阶段）。

联合稀疏形

若把k和l都写成 NAF 形式，则代表矩阵中全零列的数目就会增加，使得算法 3.48（一次处理一列）在赋值阶段的点加的数量为$5t/9$。通过对k和l采用带符号的二进制展开式，可使矩阵中全零列的数目增加。正整数k和l的联合稀疏形（JSF）代表矩阵由下列特性所表征：

1．任何连续的 3 列中至少有 1 列是零。

2．在 1 行中连续的项没有相反的符号。

3．若$k_{j+1}k_j \neq 0$，则$l_{j+1} \neq 0$且$l_j = 0$。若$l_{j+1}l_j \neq 0$，则$k_{j+1} \neq 0$且$k_j = 0$。

这种表示方法具有最小重量，在所有带符号的二进制表示中，重量定义为非零列的数目。

例 3.49　（联合稀疏形）下表给出了$k = 53$和$l = 102$的代表矩阵。

	二进制	NAF	联合稀疏形
$k=53$	0 1 1 0 1 0 1	0 1 0 −1 0 1 0 1	1 0 0 −1 0 −1 −1
$l=102$	1 1 0 0 1 1 0	1 0 −1 0 1 0 −1 0	1 1 0 1 0 −1 0
重量	6	8	5

若算法 3.48 采用 JSF，在每次迭代中处理一列，则在赋值阶段需要$t/2$次加（而不是 NAF 方法中的$5t/9$次）。算法 3.50 对于k^1和k^2寻找联合稀疏形。虽然算法用整数形式书写，但实际上只需要简单的位运算。例如，对于模 8 运算的值只需要考虑 3 位，而$\lfloor k^i/2 \rfloor$运算可丢弃最低位。

算法 3.50　联合稀疏形

输入：非负整数k^1和k^2，两者不同时为零。

输出：JSF(k^2, k^1)，k^1和k^2的联合稀疏形。

1. $l \leftarrow 0, d_1 \leftarrow 0, d_2 \leftarrow 0$。
2. 当（$k^1 + d_1 > 0$或$k^2 + d_2 > 0$）时，执行
 2.1 $\ell_1 \leftarrow d_1 + k^1, \ell_2 \leftarrow d_2 + k^2$。
 2.2 对于i从1到2，重复执行
 若ℓ_i是偶数，则$u \leftarrow 0$。
 否则
 $u \leftarrow \ell_i \bmod 4$。
 若$\ell_i \equiv \pm 3 \pmod 8$且$\ell_{3-i} \equiv 2 \pmod 4$，则$u \leftarrow -u$。
 $k_l^i \leftarrow u$。
 2.3 对于i从1到2，重复执行
 若$2d_i = 1 + k_l^i$，则$d_i \leftarrow 1 - d_i$。
 $k^i \leftarrow \lfloor k^i/2 \rfloor$。
 2.4 $l \leftarrow l + 1$。
3. 返回（$\mathrm{JSF}(k^2, k^1)$）$= \begin{pmatrix} k_{l-1}^1, \ldots, k_0^1 \\ k_{l-1}^2, \ldots, k_0^2 \end{pmatrix}$。

交错

同时计算的方法和comb方法都是利用点组合的预计算来处理多点乘。粗略地说，若每个预计算的值仅涉及一个点，则称该方法为交错。

在对点P_j和整数k^j计算$\sum k^j P_j$的过程中，交错技术计算每个$k^j P_j$时采用不同的方法，从而使计算倍点的步骤能联合进行。例如，宽度w的NAF方法对于不同的宽度都能应用，或某些点乘可以用comb方法完成。然而，计算倍点的成本取决于为了计算$k^j P_j$所需的最大倍点数，并且comb方法的优点在交错中丧失了。

算法3.51是计算$\sum_{j=1}^{v} k^j P_j$的一种交错方法，其中宽度w_j的NAF被用于k^j。对于奇数$i < 2^{w_j - 1}$，点iP_j在预计算阶段进行计算。扩展的$\mathrm{NAF}_{w_j}(k^j)$被联合处理，从左到右，每一段用一个倍点累加器。图3.8给出的是$v = 2$的情况。算法的期望运行时间近似为

$$\left\lceil |\{j : w_j > 2\}| D + \sum_{j=1}^{v} (2^{w_j - 2} - 1) A \right\rceil + \left\lceil \max_{1 \leqslant j \leqslant v} l_j D + \sum_{j=1}^{v} \frac{l_j}{w_j + 1} A \right\rceil, \qquad (3.31)$$

其中l_j表示$\mathrm{NAF}_{w_j}(k^j)$的长度，并且需要存储$\sum_{j=1}^{v} 2^{w_j - 2}$个点。

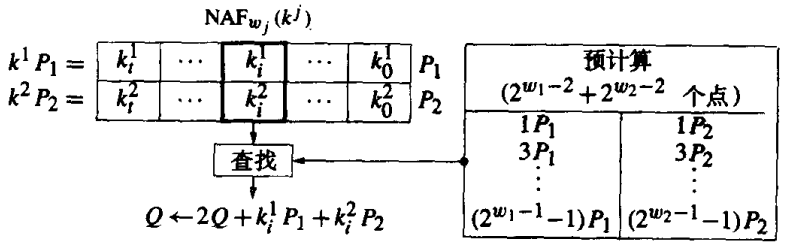


图 3.8　用交错 NAF 方法计算 $k^1P_1+k^2P_2$。对于 $v=2$ 个点示出了点乘的累加步骤。标量 k^j 被写为宽度为 w_j 的 NAF 形式

算法 3.51　交错 NAF 方法

输入：v，整数 k^j，宽度 w_j，点 P_j，$1\leqslant j\leqslant v$。

输出：$\sum_{j=1}^{v}k^jP_j$。

1. 对于 $i\in\{1,3,\ldots,2^{w_j-1}-1\}$，计算 iP_j，$1\leqslant j\leqslant v$。
2. 利用算法 3.30 计算 $\mathrm{NAF}_{w_j}(k^j)=\sum_{i=0}^{l_j-1}k_i^j2^i$，$1\leqslant j\leqslant v$。
3. 令 $l=\max\{l_j:1\leqslant j\leqslant v\}$。
4. 定义 $k_i^j=0$，$l_j\leqslant i<l$，$1\leqslant j\leqslant v$。
5. $Q\leftarrow\infty$。
6. 对于 i 从 $l-1$ 到 0，重复执行
 6.1 $Q\leftarrow 2Q$。
 6.2 对于 j 从 1 到 v，重复执行
 若 $k_i^j\neq 0$，则
 若 $k_i^j>0$，则 $Q\leftarrow Q+k_i^jP_j$。
 否则，$Q\leftarrow Q-k_i^jP_j$。
7. 返回 (Q)。

注 3.52（与同时方法的比较）考虑计算 $kP+lQ$，其中 k 和 l 的长度近似相等。同时滑动方法和交错方法本质上需要相同的倍点次数，而与窗口的宽度无关。对于给定的 w，同时滑动方法需要存储 $3\cdot 2^{2(w-1)}$ 个点，在赋值阶段需要近似 $t/(w+(1/3))$ 次点加。而交错方法对于 k 上的宽度 $2w+1$ 和 l 上的宽度 $2w$ 时，需要同样数量的存储，但是在赋值阶段只需要 $(4w+3)t/(4w^2+5w+2)<t/(w+(1/2))$ 次加法。在预计算时使用交错方法也是很好的，因为涉及已知点 P 的运算可以在前面完成（建议应用大宽度的 $\mathrm{NAF}_w(k)$）。这与在同时方法中需要联合计算不同。表 3.6 给出了在 P 点已知（Q 点未知）的情况下各种方法计算 $kP+lQ$

需要的计算量。

在预计算所需的存储限于4个点（包括P和Q）的情况下，宽度为3的NAF的交错或采用JSF本质上具有相同的性能，在预计算阶段交错需要1个或2个以上的倍点计算。表3.6给出了小窗口情况下的性能比较。

表3.6 计算$kP+lQ$需要的计算量，其中k和l是t位整数。涉及P点的计算排除在外

方　法	w	存　储	点　加	倍　点
算法3.48	1	3	$1+3t/4\approx 1+0.75t$	t
算法3.48	2	15	$9+15t/32\approx 9+0.47t$	$2+t$
算法3.48带滑动	2	12	$9+3t/7\approx 9+0.43t$	$2+t$
算法3.48带NAF		4	$2+5t/9\approx 2+0.56t$	t
算法3.48带JSF		4	$2+t/2\approx 2+0.5t$	t
3-NAF交错	3, 3	2+2	$1+t/2\approx 1+0.5t$	$1+t$
5-NAF且4-NAF交错	5, 4	8+4	$3+11t/30\approx 3+0.37t$	$1+t$

为计算kP，可以考虑用交错取代comb方法（算法3.44）。在这种情况下，k的代表矩阵用交错方法（算法3.51）进行处理，k^j由$k=\sum_{j=1}^{w}k^j 2^{(j-1)d}$给出，点$P_j$由$P_j=2^{(j-1)d}P$给出，$1\leqslant j\leqslant w$，$d$由算法3.44所定义。表3.7对于固定存储比较了comb方法和交错方法。

表3.7 对于固定存储，comb方法和交错方法计算kP（P点已知）所需的近似计算量。k的位长用t表示。对于交错方法，列出了在计算NAF时所用的每行宽度

方　法	行	存　储	点　加	倍　点
comb	2	3	$3t/8\approx 0.38t$	$t/2$
交错（3, 3）	2	4	$t/4\approx 0.25t$	$t/2$
comb	4	15	$15t/64\approx 0.23t$	$t/4$
comb（两个表）	3	14	$7t/24\approx 0.29t$	$t/6$
交错（4, 4, 4, 4）	4	16	$t/4\approx 0.25t$	$t/4$
交错（4, 4, 4, 3, 3）	5	16	$11t/50\approx 0.22t$	$t/5$
comb	5	31	$31t/160\approx 0.19t$	$t/5$
comb（两个表）	4	30	$15t/64\approx 0.23t$	$t/8$
交错（5, 5, 5, 4, 4）	5	32	$9t/50\approx 0.18t$	$t/5$
交错（5, 5, 4, 4, 4, 4）	6	32	$17t/90\approx 0.19t$	$t/6$

3.4 Koblitz曲线

Koblitz曲线也称为反常二进制曲线，是一种定义在$\mathbb{F}_2$上的椭圆曲线。这种曲线的主要

优点是点乘算法可以不使用倍点运算。本节所讨论的内容主要依据 Solinas 的论文[446]，该论文中提供了事实证明和算法分析。

定义 3.53　Koblitz 曲线是定义在$\mathbb{F}_2$上的下列椭圆曲线：

$$E_0 : y^2 + xy = x^3 + 1,$$
$$E_0 : y^2 + xy = x^3 + 1。$$

在密码协议中，人们使用扩域$\mathbb{F}_{2^m}$上的椭圆曲线有理数点群 $E_0(\mathbb{F}_{2^m})$和 $E_1(\mathbb{F}_{2^m})$。令 $a \in \{0,1\}$，对于 m的每一个因子 l，$E_a(\mathbb{F}_{2^l})$是 $E_a(\mathbb{F}_{2^m})$ 的子群。因此$\#E_a(\mathbb{F}_{2^l})$能整除 $\#E_a(\mathbb{F}_{2^m})$。具体地，因为 $\#E_0(\mathbb{F}_2) = 4$，$\#E_1(\mathbb{F}_2) = 2$，所以$\#E_0(\mathbb{F}_{2^m})$是4的倍数而且 $\#E_1(\mathbb{F}_{2^m})$是 2 的倍数。

定义 3.54　Koblitz 曲线 E_a在$\mathbb{F}_{2^m}$上有殆素群阶，若 $\#E_a(\mathbb{F}_{2^m}) = hn$，其中$n$是素数并且

$$h = \begin{cases} 4, & \text{若 } a = 0 \\ 2, & \text{若 } a = 1。 \end{cases}$$

h称为余因子。

我们将假设在本节其余部分内 E_a为具有殆素群阶$\#E_a(\mathbb{F}_{2^m})$的 Koblitz 曲线。注意到，若m是素数，则$\#E_a(\mathbb{F}_{2^m})$只能是殆素数的。利用定理 3.11 可以有效地计算群阶$\#E_a(\mathbb{F}_{2^m})$。表 3.8 列出了扩张度$m \in [100, 600]$和 Koblitz 曲线 E_a，其中$\#E_a(\mathbb{F}_{2^m})$是殆素数的。

3.4.1　Frobenius 映射和环$\mathbb{Z}[\tau]$

定义 3.55　令 E_a是 Koblitz 曲线。定义 Frobenius 映射$\tau : E_a(\mathbb{F}_{2^m}) \to E_a(\mathbb{F}_{2^m})$为

$$\tau(\infty) = \infty, \quad \tau(x, y) = (x^2, y^2)。$$

Frobenius 映射能够有效地进行计算，因为域$\mathbb{F}_{2^m}$上的平方相对比较容易计算（见 2.3.4 节）。众所周知，

$$(\tau^2 + 2)P = \mu\tau(P)，\text{对于所有 } P \in E_a(\mathbb{F}_{2^m}),$$

其中$\mu = (-1)^{1-a}$且$\tau^l(P)$表示 l 次将映射 τ 作用于点 P。因此 Frobenius 映射可以被认为是一个满足

$$\tau^2 + 2 = \mu\tau \tag{3.32}$$

的复数，其中$\tau = (\mu + \sqrt{-7})/2$。令$\mathbb{Z}[\tau]$表示整数系数的 τ 的多项式环。若$u_{l-1}\tau^{l-1} + \cdots + u_1\tau + u_0 \in \mathbb{Z}[\tau]$且$P \in E_a(\mathbb{F}_{2^m})$，则

$$(u_{l-1}\tau^{l-1}+\cdots+u_1\tau+u_0)P=u_{l-1}\tau^{l-1}(P)+\cdots+u_1\tau(P)+u_0P。\qquad(3.33)$$

表 3.8 具有殆素群阶$\#E_a(\mathbb{F}_{2^m})$且$m\in[100,600]$的 Koblitz 曲线E_a

m	曲 线	$\#E_a(\mathbb{F}_{2^m})$的素因子
101	E_1	$2\cdot 1267650600228230886142808508011$
103	E_0	$2^2\cdot 2535301200456459535862530067069$
107	E_0	$2^2\cdot 40564819207303335604363489037809$
107	E_1	$2\cdot 81129638414606692182851032212511$
109	E_1	$2\cdot 324518553658426701487448656461467$
113	E_1	$2\cdot 5192296858534827627896703833467507$
131	E_0	$2^2\cdot 6805647338418769269323201294934099851 29$
163	E_1	$2\cdot 584600654932361167281474175359844834832911 8574063$
233	E_0	$2^2\cdot 345087317339528189371737793113851276057094098886225212632808 7024741343$
239	E_0	$2^2\cdot 220855883097298041197912187592864814948216561321709848887480219215362213$
277	E_0	$2^2\cdot 607084028820540334662331845882349658325751104987865087648841755618916 22165064650683$
283	E_0	$2^2\cdot 3885337784451458141838923813647037813284811733793061324295874997529815829704422603873$
283	E_1	$2\cdot 777067556890291628367784762729407562656963124483099352142274928285160 2622232822777663$
311	E_1	$2\cdot 208592483976651375233888838493120323691670363507171116673989121858491 635472665429482533830218 3$
331	E_1	$2\cdot 218725072478301192437250222711762136535316943089322764344701030671135871258677658859434350525561430 3$
347	E_1	$2\cdot 143343663499379469475676305956380433799785311823017565728537420307240 763803325774115493723193900257029311$
349	E_1	$2^2\cdot 2866873269987589389513526119127608675995706236460351478840674433541530787625118990359606515490187750443 23$
359	E_1	$2\cdot 587135645693458306972370149197334256843920637227079966811081824609485 917244124494882365172478748165648998663$
409	E_0	$2^2\cdot 330527984395124299475957654016385519914202341482140609642324395022880 711289249191050673258457777458014096366590617731358671$
571	E_0	$2^2\cdot 193226876150862917234767594546599367214946366485321749932861762572575 957114478021226813397852270671183470671280082535146127367497406661731 192968242161709250355573368527667 3$

Koblitz 曲线的有效点乘算法可以采用如下策略：对于给定的整数 k 和 k 的“好”的表示式 $k=\sum_{i=0}^{l-1}u_i\tau^i$，可以利用式(3.33)来计算 kP。其中“好”的表示式意味着 l 相对小且非零数字 u_i 也小（如±1）且稀疏。

因为 $\tau^2=\mu\tau-2$，所以 $\mathbb{Z}[\tau]$ 中的每个元素 α 都可表示为规范式 $\alpha=a_0+a_1\tau$，其中 $a_0,a_1\in\mathbb{Z}$。

定义 3.56　$\alpha=a_0+a_1\tau\in\mathbb{Z}[\tau]$的范数是 α 和它的复共轭的乘积，即

$$N(a_0+a_1\tau)=a_0^2+\mu a_0a_1+2a_1^2。$$

定理 3.57（*规范函数的性质*）

(i) 对于所有的 $\alpha\in\mathbb{Z}[\tau]$，$N(\alpha)\geqslant 0$，当且仅当 $\alpha=0$ 时取等号。

(ii) 1 和–1 是 $\mathbb{Z}[\tau]$ 中范数为 1 的仅有的元素。

(iii) $N(\tau)=2$ 且 $N(\tau-1)=h$。

(iv) $N(\tau^m-1)=\#E_a(\mathbb{F}_{2^m})$，且 $N((\tau^m-1)/(\tau-1))=n$。

(v) 规范函数具有相乘性，即对于所有的 $\alpha_1,\alpha_2\in\mathbb{Z}[\tau]$，$N(\alpha_1\alpha_2)=N(\alpha_1)N(\alpha_2)$。

(vi) 对于规范函数来说 $\mathbb{Z}[\tau]$ 是 Euclidean 集合。即对于任意的 $\alpha,\beta\in\mathbb{Z}[\tau]$，$\beta\neq 0$，存在 $\kappa,\rho\in\mathbb{Z}[\tau]$（不一定惟一），使得 $\alpha=\kappa\beta+\rho$ 且 $N(\rho)<N(\beta)$。

τ-adic 非邻接形（TNAF）

根据定理 3.57 可以得出，任意正整数 k 可以写成如下形式：$k=\sum_{i=0}^{l-1}u_i\tau^i$，其中每个 $u_i\in\{0,\pm1\}$。通过用 τ 连续地除 k 可以得到 τ-adic 表达式，其中数字 u_i 是每一步除法的余数。这一过程类似于用 2 连续地除余数来获得二进制表达式。为了减少式(3.33)中的点加的数量，希望获得 k 的只有少量非零数字的 τ-adic 表达式。采用 τ-adic NAF 可以实现这一点，τ-adic NAF 与二元 NAF 类似（参见定义 3.28）。

定义 3.58　一个非零元素 $\kappa\in\mathbb{Z}[\tau]$ 的 τ-adic NAF 或 TNAF 是表达式 $\kappa=\sum_{i=0}^{l-1}u_i\tau^i$，其中每个 $u_i\in\{0,\pm1\}$ 且 $u_{l-1}\neq 0$，而且没有两个连续的数字 u_i 为非零。TNAF 的长度为 l。

定理 3.59（TNAF 的性质）令 $\kappa\in\mathbb{Z}[\tau],\kappa\neq 0$。

(i) κ 有惟一的 TNAF，记为 TNAF(κ)。

(ii) 若 TNAF(κ)的长度 $l(\kappa)$ 大于 30，则

$$\mathrm{lb}(N(\kappa))-0.55<l(\kappa)<\mathrm{lb}(N(\kappa))+3.52。$$

(iii) 所有长度为 l 的 TNAF 中非零数字的平均密度近似为 1/3。

用算法 3.61 可以有效地计算 TNAF (κ)。算法 3.61 与算法 3.30 类似。TNAF (κ)的各数字通过如下方法得到：用 τ 连续除 κ，并取为 0 或±1 的余数。若 κ 不能整除 τ，则选择余数 $r \in \{-1, 1\}$，以使商 $(\kappa - r)/\tau$ 能整除 τ，确保下一个 TNAF 的数字为 0。利用下列结果容易完成用 τ 和 τ^2 除 α 的运算，其中 $\alpha \in \mathbb{Z}[\tau]$。

定理 3.60 （$\mathbb{Z}[\tau]$中的用 τ 和 τ^2 的除法）令 $\alpha = r_0 + r_1\tau \in \mathbb{Z}[\tau]$。

(i) 当且仅当 r_0 是偶数时，α 整除 τ。若 r_0 是偶数，则

$$\alpha/\tau = (r_1 + \mu r_0/2) - (r_0/2)\tau。$$

(ii) 当且仅当 $r_0 \equiv 2r_1 \pmod 4$ 时，α 整除 τ^2。

算法 3.61 计算 $\mathbb{Z}[\tau]$中一个元素的 TNAF

输入：$\kappa = r_0 + r_1\tau \in \mathbb{Z}[\tau]$。

输出：TNAF(κ)。

1. $i \leftarrow 0$。
2. 当 $r_0 \neq 0$ 或 $r_1 \neq 0$ 时，重复执行
 2.1 若 r_0 是奇数，则 $u_i \leftarrow 2 - (r_0 - 2r_1 \bmod 4)$，$r_0 \leftarrow r_0 - u_i$。
 2.2 否则，$u_i \leftarrow 0$。
 2.3 $t \leftarrow r_0, r_0 \leftarrow r_1 + \mu r_0/2, r_1 \leftarrow -t/2, i \leftarrow i + 1$。
3. 返回 $(u_{i-1}, u_{i-2}, \ldots, u_1, u_0)$。

为了计算 kP，可以首先利用算法 3.61 得到 TNAF(k)，然后利用式(3.33)进行计算。根据定理 3.59 (ii)，TNAF(k)的长度近似为 $\mathrm{lb}\,(N(k)) = 2\,\mathrm{lb}\,k$，即两倍于 NAF($k$)的长度。为了巧妙地解决 TNAF 的长度问题，要注意若 $\gamma \equiv k \pmod{\tau^m - 1}$，则对于所有的 $P \in E_a(\mathbb{F}_{2^m})$，都有 $kP = \gamma P$。这是因为

$$(\tau^m - 1)(P) = \tau^m(P) - P = P - P = \infty。$$

此外还表明，若 $\rho \equiv k \pmod \delta$，其中 $\delta = (\tau^m - 1)/(\tau - 1)$，则对于 $E_a(\mathbb{F}_{2^m})$中所有阶为 n 的点 P 都有 $kP = \rho P$。现在的策略是寻找出范数尽可能小且有 $\rho \equiv k \pmod \delta$ 的 $\rho \in \mathbb{Z}[\tau]$，然后应用 TNAF($\rho$)来计算 ρP。对于任何 α，$\beta \in \mathbb{Z}[\tau]$（$\beta \neq 0$），算法 3.62 找出商 $\kappa \in \mathbb{Z}[\tau]$ 和余数 $\rho \in \mathbb{Z}[\tau]$，其中 $\alpha = \kappa\beta + \rho$ 且 $N(\rho)$尽可能小。作为一个子程序，算法 3.63 用来寻找 $\mathbb{Z}[\tau]$中接近于一个给定的复数 $\lambda_0 + \lambda_1\tau$ 的元素，其中 λ_1 和 $\lambda_0 \in \mathbb{Q}$。

算法 3.62　$\mathbb{Z}[\tau]$的除法

输入：$\alpha=a_0+a_1\tau\in\mathbb{Z}[\tau],\beta=b_0+b_1\tau\in\mathbb{Z}[\tau]$，$\beta\neq 0$。

输出：$\kappa=q_0+q_1\tau,\rho=r_0+r_1\tau\in\mathbb{Z}[\tau]$，$\alpha=\kappa\beta+\rho$且$N(\rho)\leqslant\frac{4}{7}N(\beta)$。

1. $g_0\leftarrow a_0b_0+\mu a_0b_1+2a_1b_1$。
2. $g_1\leftarrow a_1b_0-a_0b_1$。
3. $N\leftarrow b_0^2+\mu b_0b_1+2b_1^2$。
4. $\lambda_0\leftarrow g_0/N$，$\lambda_1\leftarrow g_1/N$。
5. 应用算法 3.63 计算$(q_0,q_1)\leftarrow$Round (λ_0,λ_1)。
6. $r_0\leftarrow a_0-b_0q_0+2b_1q_1$。
7. $r_1\leftarrow a_1-b_1q_0-b_0q_1-\mu b_1q_1$。
8. $\kappa\leftarrow q_0+q_1\tau$。
9. $\rho\leftarrow r_0+r_1\tau$。
10. 返回(κ,ρ)。

算法 3.63　$\mathbb{Z}[\tau]$中的循环

输入：自然数λ_0和λ_1。

输出：整数q_0,q_1使得$q_0+q_1\tau$接近于复数$\lambda_0+\lambda_1\tau$。

1. 对于i从 0 到 1，执行

 1.1 $f_i\leftarrow\lfloor\lambda_i+\frac{1}{2}\rfloor,\eta_i\leftarrow\lambda_i-f_i,h_i\leftarrow 0$。
2. $\eta\leftarrow 2\eta_0+\mu\eta_1$。
3. 若$\eta\geqslant 1$，则

 3.1 若$\eta_0-3\mu\eta_1<-1$，则$h_1\leftarrow\mu$；否则，$h_0\leftarrow 1$。

 否则

 3.2 若$\eta_0+4\mu\eta_1\geqslant 2$，则$h_1\leftarrow\mu$。
4. 若$\eta<-1$，则

 4.1 若$\eta_0-3\mu\eta_1\geqslant 1$，则$h_1\leftarrow-\mu$；否则，$h_0\leftarrow-1$。

 否则

 4.2 若$\eta_0+4\mu\eta_1<-2$，则$h_1\leftarrow-\mu$。
5. $q_0\leftarrow f_0+h_0,q_1\leftarrow f_1+h_1$。
6. 返回(q_0,q_1)。

定义 3.64 令$\alpha,\beta\in\mathbb{Z}[\tau]$，$\beta\neq 0$，则$\alpha \bmod \beta$定义为算法 3.62 的输出$\rho\in\mathbb{Z}[\tau]$。

在某些平台上实现算法 3.62 以计算$\rho=k \bmod \delta$是很麻烦的，因为需要两个多精度整数的除法（在步骤 4）。而用算法 3.65 计算一个元素$\rho'\equiv k \pmod{\delta}$不需要复杂的多精度整数的除法。我们记$\rho'=k \text{ part mod } \delta$。Solinas 证明了若$l(\rho)\leqslant m+a$且$C\geqslant 2$，则$l(\rho')\leqslant m+a+3$。然而，$l(\rho')$明显大于$l(\rho)$仍是可能的。但在实际上并不需要关注此问题，因为$\rho'\neq\rho$的概率小于$2^{-(C-5)}$，因此选择足够大的$C$可确保$\rho'=\rho$的概率足够大。

算法 3.65 模$\delta=(\tau^m-1)/(\tau-1)$的部分约减

输入：$k\in[1,n-1]$, $C\geqslant 2$, $s_0=d_0+\mu d_1$, $s_1=-d_1$，其中$\delta=d_0+d_1\tau$。

输出：$\rho'=k \text{ partmod } \delta$。

1. $k'\leftarrow\lfloor k/2^{a-C+(m-9)/2}\rfloor$。
2. $V_m\leftarrow 2^m+1-\#E_a(\mathbb{F}_{2^m})$。
3. 对于i从 0 到 1，执行
 3.1 $g'\leftarrow s_i\cdot k'$. $j'\leftarrow V_m\cdot\lfloor g'/2^m\rfloor$。
 3.2 $\lambda_i\leftarrow\lfloor(g'+j')/2^{(m+5)/2}+\frac{1}{2}\rfloor/2^C$。
4. 利用算法 3.63 计算$(q_0,q_1)\leftarrow \text{Round}(\lambda_0,\lambda_1)$。
5. $r_0\leftarrow k-(s_0+\mu s_1)q_0-2s_1q_1$, $r_1\leftarrow s_1q_0-s_0q_1$。
6. 返回$(r_0+r_1\tau)$。

3.4.2 点乘

算法3.66 是一种有效的点乘方法，它综合采用了前面的各种思想。因为 TNAF(ρ')的长度近似为m，且它的密度的期望值约为 1/3，因此算法 3.66 的期望运行时间近似为

$$\frac{m}{3}A。 \tag{3.34}$$

算法 3.66 Koblitz 曲线点乘的 TNAF 方法

输入：整数$k\in[1,n-1]$, $P\in E(\mathbb{F}_{2^m})$，P的阶为n。

输出：kP。

1. 利用算法 3.65 计算$\rho'=k \text{ partmod } \delta$。
2. 利用算法 3.61 计算 TNAF$(\rho')=\sum_{i=0}^{l-1}u_i\tau^i$。
3. $Q\leftarrow\infty$。

4. 对于 i 从 $l-1$ 到 0，重复执行
 4.1 $Q \leftarrow \tau Q$。
 4.2 若 $u_i = 1$，则 $Q \leftarrow Q + P$。
 4.3 若 $u_i = -1$，则 $Q \leftarrow Q - P$。
5. 返回(Q)。

窗口方法

若有额外的存储器可用，则通过采用窗口方法，一次处理 ρ' 的 w 位，可使算法 3.66 的运算加速。这是通过采用宽度 w 的 TNAF 方法取得的，这一方法可以看做是二进制宽度 w 的 NAF 方法的 τ-adic 方法（见定义 3.32）。

定理 3.67　令 $\{U_k\}$ 是如下定义的整数序列：$U_0 = 0, U_1 = 1$，对于所有的 $k \geqslant 1$，$U_{k+1} = \mu U_k - 2U_{k-1}$。

(i) 对于所有的 $k \geqslant 1$，$U_k^2 - \mu U_{k-1}U_k + 2U_{k-1}^2 = 2^{k-1}$。

(ii) 对于所有的 $k \geqslant 1$，令 $t_k = 2U_{k-1}U_k^{-1} \bmod 2^k$（因为 U_k 是奇数，并且对于所有的 $k \geqslant 1$，$U_k^{-1} \bmod 2^k$ 确实存在）。于是对于所有的 $k \geqslant 1$，都有 $t_k^2 + 2 \equiv \mu t_k \pmod{2^k}$。

根据式(3.32)和定理 3.67(ii)可知，由 $\tau \mapsto t_w$ 诱导出的映射 $\phi_w : \mathbb{Z}[\tau] \to \mathbb{Z}_{2^w}$ 是一个带有核 $\{\alpha \in \mathbb{Z}[\tau] : \tau^w$ 除 $\alpha\}$ 的满射环同态。此外，$\mathbb{Z}[\tau] \bmod \tau^w$ 的等价类的不同代表元的集合是 $\{0, \pm 1, \pm 2, \pm 3, \ldots, \pm(2^{w-1} - 1), -2^{w-1}\}$，而 $\{\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$ 都不能整除 τ。

定义 3.68　令 $w \geqslant 2$ 是一个正整数。定义 $\alpha_i = i \bmod \tau^w$，对于 $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$。一个非零元素 $\kappa \in \mathbb{Z}[\tau]$ 的宽度 w 的 TNAF 是表达式 $\kappa = \sum_{i=0}^{l-1} u_i \tau^i$，其中 $u_i \in \{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$，$u_{l-1} \neq 0$，且任何连续 w 位数字中最多只有 1 位不为 0。称宽度 w 的 TNAF 的长度为 l。

注意，$\mathrm{TNAF}_2(\kappa) = \mathrm{TNAF}(\kappa)$。表 3.9 和表 3.10 列出对于 $a \in \{0, 1\}$ 且 $3 \leqslant w \leqslant 6$ 的 α_u。对于每个 α_u，所给出的表达式最多有两项涉及 τ 的幂和其他 α_u。$\mathrm{TNAF}(\alpha_u) = (u_{l-1}, \ldots, u_1, u_0)$ 意味着 $\sum_{i=0}^{l-1} u_i \tau^i$。表中最后一列的多数元素是由 TNAF 得到的，但也有少数例外，因为在用 TNAF 方法时效率不高。对于这些表达式，每个 $\alpha_u P$ 最多用一个椭圆曲线加法就可计算。

利用算法 3.69 可以有效地计算 $\mathrm{TNAF}_w(\rho)$。在算法3.69 中，$k \bmod 2^w$ 表示满足 $u \equiv k \pmod{2^w}$ 且 $-2^{w-1} \leqslant u < 2^{w-1}$ 的整数 u。$\mathrm{TNAF}_w(\rho)$ 的各位数字可通过如下方法来获

得：连续地用 τ 去除 ρ 并取余数 $\gamma \in \{0, \pm\alpha_1, \pm\alpha_3 \ldots, \pm\alpha_{2^{w-1}-1}\}$。如果 ρ 不能整除 τ 且余数选择为 α_u，其中 $u = \phi_w(\rho) \bmod 2^w$，则 $(\rho - \alpha_u)/\tau$ 将整除 τ^{w-1}，从而确保后面的 $w-1$ 位数字都是 0。

表 3.9 $a=0$ 且 $3 \leqslant w \leqslant 6$ 时 $\alpha_u = u \bmod \tau^w$ 的表达式

w	u	$u \bmod \tau^w$	TNAF($u \bmod \tau^w$)	α_u
3	1	1	(1)	1
	3	$\tau+1$	$(-1,0,-1)$	$\tau+1$
4	1	1	(1)	1
	3	$-\tau-3$	$(1,0,-1)$	τ^2-1
	5	$-\tau-1$	$(1,0,1)$	τ^2+1
	7	$-\tau+1$	$(1,0,0,-1)$	τ^3-1
5	1	1	(1)	1
	3	$-\tau-3$	$(1,0,-1)$	τ^2-1
	5	$-\tau-1$	$(1,0,1)$	τ^2+1
	7	$-\tau+1$	$(1,0,0,-1)$	τ^3-1
	9	$-2\tau-3$	$(1,0,1,0,0,1)$	$\tau^3\alpha_5+1$
	11	$-2\tau-1$	$(-1,0,-1,0,-1)$	$-\tau^2\alpha_5-1$
	13	$-2\tau+1$	$(-1,0,-1,0,1)$	$-\tau^2\alpha_5+1$
	15	$3\tau+1$	$(1,0,0,0,-1)$	$\tau^2\alpha_5-\alpha_5$
6	1	1	(1)	1
	3	3	$(-1,0,0,1,0,-1)$	$\tau^2\alpha_{25}-1$
	5	5	$(-1,0,0,1,0,1)$	$\tau^2\alpha_{25}+1$
	7	$-2\tau-5$	$(1,0,1,0,0,-1)$	$-\tau^3\alpha_{27}-1$
	9	$-2\tau-3$	$(1,0,1,0,0,1)$	$-\tau^3\alpha_{27}+1$
	11	$-2\tau-1$	$(-1,0,-1,0,-1)$	$\tau^2\alpha_{27}-1$
	13	$-2\tau+1$	$(-1,0,-1,0,1)$	$\tau^2\alpha_{27}+1$
	15	$3\tau+1$	$(1,0,0,0,-1)$	$-\tau^2\alpha_{27}+\alpha_{27}$
	17	$3\tau+3$	$(1,0,0,0,1)$	$-\tau^2\alpha_{27}+\alpha_{29}$
	19	$3\tau+5$	$(1,0,0,-1,0,1,0,-1)$	$-\tau^2\alpha_3-1$
	21	$-4\tau-3$	$(-1,0,1,0,1)$	$\tau^2\alpha_{29}+1$
	23	$\tau-3$	$(-1,0,0,-1)$	$-\tau^3-1$
	25	$\tau-1$	$(-1,0,0,1)$	$-\tau^3+1$
	27	$\tau+1$	$(-1,0,-1)$	$-\tau^2-1$
	29	$\tau+3$	$(-1,0,1)$	$-\tau^2+1$
	31	$\tau+5$	$(-1,0,0,0,0,-1)$	$\tau^2\alpha_{25}+\alpha_{27}$

表 3.10　$a=1$ 且 $3\leqslant w\leqslant 6$ 时 $\alpha_u=u \bmod \tau^w$ 的表达式

w	u	$u \bmod \tau^w$	TNAF($u \bmod \tau^w$)	α_u
3	1	1	(1)	1
	3	$-\tau+1$	$(-1,0,-1)$	$-\tau+1$
4	1	1	(1)	1
	3	$\tau-3$	$(1,0,-1)$	τ^2-1
	5	$\tau-1$	$(1,0,1)$	τ^2+1
	7	$\tau+1$	$(-1,0,0,-1)$	$-\tau^3-1$
5	1	1	(1)	1
	3	$\tau-3$	$(1,0,-1)$	τ^2-1
	5	$\tau-1$	$(1,0,1)$	τ^2+1
	7	$\tau+1$	$(-1,0,0,-1)$	$-\tau^3-1$
	9	$2\tau-3$	$(-1,0,-1,0,0,1)$	$-\tau^3\alpha_5+1$
	11	$2\tau-1$	$(-1,0,-1,0,-1)$	$-\tau^2\alpha_5-1$
	13	$2\tau+1$	$(-1,0,-1,0,1)$	$-\tau^2\alpha_5+1$
	15	$-3\tau+1$	$(1,0,0,0,-1)$	$\tau^2\alpha_5-\alpha_5$
6	1	1	(1)	1
	3	3	$(1,0,0,1,0,-1)$	$\tau^2\alpha_{25}-1$
	5	5	$(1,0,0,1,0,1)$	$\tau^2\alpha_{25}+1$
	7	$2\tau-5$	$(-1,0,-1,0,0,-1)$	$\tau^3\alpha_{27}-1$
	9	$2\tau-3$	$(-1,0,-1,0,0,1)$	$\tau^3\alpha_{27}+1$
	11	$2\tau-1$	$(-1,0,-1,0,-1)$	$\tau^2\alpha_{27}-1$
	13	$2\tau+1$	$(-1,0,-1,0,1)$	$\tau^2\alpha_{27}+1$
	15	$-3\tau+1$	$(1,0,0,0,-1)$	$-\tau^2\alpha_{27}+\alpha_{27}$
	17	$-3\tau+3$	$(1,0,0,0,1)$	$-\tau^2\alpha_{27}+\alpha_{29}$
	19	$-3\tau+5$	$(-1,0,0,-1,0,1,0,-1)$	$-\tau^2\alpha_3-1$
	21	$4\tau-3$	$(-1,0,1,0,1)$	$\tau^2\alpha_{29}+1$
	23	$-\tau-3$	$(1,0,0,-1)$	τ^3-1
	25	$-\tau-1$	$(1,0,0,1)$	τ^3+1
	27	$-\tau+1$	$(-1,0,-1)$	$-\tau^2-1$
	29	$-\tau+3$	$(-1,0,1)$	$-\tau^2+1$
	31	$-\tau+5$	$(1,0,0,0,0,-1)$	$\tau^2\alpha_{25}+\alpha_{27}$

算法 3.69　计算 $\mathbb{Z}[\tau]$ 中一个元素的宽度 w 的 TNAF

输入：$w, t_w, \alpha_u=\beta_u+\gamma_u\tau$，$u\in\{1,3,5,\ldots,2^{w-1}-1\}$，$\rho=r_0+r_1\tau\in\mathbb{Z}[\tau]$。

输出：$\mathrm{TNAF}_w(\rho)$。

1. $i\leftarrow 0$。

2. 当 $r_0\neq 0$ 或 $r_1\neq 0$ 时，执行

2.1 若r_0是奇数，则

$u \leftarrow r_0 + r_1 t_w \bmod 2^w$。

若$u > 0$，则$s \leftarrow 1$；否则，$s \leftarrow -1$，$u \leftarrow -u$。

$r_0 \leftarrow r_0 - s\beta_u$，$r_1 \leftarrow r_1 - s\gamma_u$，$u_i \leftarrow s\alpha_u$。

2.2 否则，$u_i \leftarrow 0$。

2.3 $t \leftarrow r_0$，$r_0 \leftarrow r_1 + \mu r_0/2$，$r_1 \leftarrow -t/2$，$i \leftarrow i+1$。

3. 返回$(u_{i-1}, u_{i-2}, \ldots, u_1, u_0)$。

算法3.70是一个有效的点乘算法，它采用了宽度w的TNAF技术。因为TNAF(ρ')的期望长度为m，而且其期望密度为$1/(w+1)$，所以算法3.70的期望运行时间近似为

$$\left(2^{w-2} - 1 + \frac{m}{w+1}\right)A。 \tag{3.35}$$

算法3.70 Koblitz曲线点乘的窗口TNAF方法

输入：窗口宽度w，整数$k \in [1, n-1]$，$P \in E(\mathbb{F}_{2^m})$，$P$的阶为$n$。

输出：kP。

1. 利用算法3.65计算$\rho' = k \ \text{partmod}\, \delta$。
2. 利用算法3.69计算$\text{TNAF}_w(\rho') = \sum_{i=0}^{l-1} u_i \tau^i$。
3. 对于$u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$，计算$P_u = \alpha_u P$。
4. $Q \leftarrow \infty$。
5. 对于i从$l-1$到0，重复执行

5.1 $Q \leftarrow \tau Q$。

5.2 若$u_i \neq 0$，则

令u满足$\alpha_u = u_i$或$\alpha_{-u} = -u_i$。

若$u > 0$，则$Q \leftarrow Q + P_u$。

否则，$Q \leftarrow Q - P_{-u}$。

6. 返回(Q)。

3.5 具有有效可计算自同态的曲线

Frobenius映射（定义3.55）是椭圆曲线的自同态的一个例子。本节给出一种加速具有有效可计算自同态的椭圆曲线点乘运算的一般技术。虽然这种技术并不产生像3.4节中的

Koblitz 曲线（那里所有的倍点运算被相当快的 Frobenius 映射所替换）那样巨大的提速，然而却可用来加速大多数曲线的点乘速度，包括某些大素域上的椭圆曲线。粗略地说，若自同态可以比小量的倍点计算得快，则这种技术大约可省掉一半的倍点运算且节省约 33% 的点乘时间。

椭圆曲线的自同态

令 E 是定义在域 K 上的椭圆曲线。E 上的点（其坐标可以在 K 的任何扩域上）的集合也用 E 来表示。域 K 上的 E 的一个自同态 ϕ 是一个映射 $\phi: E \to E$，使得 $\phi(\infty)=\infty$ 和 $\phi(P)=(g(P),h(P))$，对于所有的 $P \in E$，其中 g 和 h 是系数取自 K 中的有理函数。K 上 E 的所有自同态构成一个环，称为 K 上 E 的自同态环。一个自同态 ϕ 还是一个群同态，即

$$\phi(P_1+P_2)=\phi(P_1)+\phi(P_2)，\text{对于所有的} P_1, P_2 \in E。$$

自同态 ϕ 的特征多项式是 $\mathbb{Z}[X]$ 中次数最低的首一多项式 $f(X)$，且 $f(\phi)=0$，即对于所有的 $P \in E$ 都有 $f(\phi)(P)=\infty$。若 E 是非超奇异椭圆曲线，则 ϕ 的特征多项式的次数为 1 或 2。

例 3.71　（*椭圆曲线的自同态*）

(i) 令 E 是 $\mathbb{F}_q$ 上的椭圆曲线，对于每个整数 m，由

$$[m]: P \mapsto mP$$

所定义的用 m 乘的映射 $[m]: E \to E$ 是 $\mathbb{F}_q$ 上的椭圆曲线 E 的一个自同态。一个特例是负映射 $P \mapsto -P$。映射 $[m]$ 的特征多项式是 $X-m$。

(ii) 令 E 是 $\mathbb{F}_q$ 上的椭圆曲线，则由

$$\phi: (x,y) \mapsto (x^q, y^q), \quad \phi: \infty \mapsto \infty$$

所定义的 q 次幂映射 $\phi: E \to E$ 是 $\mathbb{F}_q$ 上的椭圆曲线 E 的一个自同态，称为 Frobenius 自同态。ϕ 的特征多项式是 X^2-tX+q，其中 $t=q+1-\#E(\mathbb{F}_q)$。

(iii) 令 p 是一个素数，$p \equiv 1 \pmod 4$。考虑定义在 $\mathbb{F}_p$ 上的椭圆曲线

$$E: y^2=x^3+ax。$$

令 $i \in \mathbb{F}_p$ 是一个阶为 4 的元素。则由

$$\phi: (x,y) \mapsto (-x, iy), \quad \phi: \infty \mapsto \infty$$

所定义的映射 $\phi: E \to E$ 是 $\mathbb{F}_p$ 上的椭圆曲线 E 的一个自同态。注意，$\phi(P)$ 仅用一次乘就可以计算。ϕ 的特征多项式是 X^2+1。

(iv) 令 p 是一个素数，$p \equiv 1 \pmod 3$，考虑定义在 $\mathbb{F}_p$ 上的椭圆曲线

$$E: y^2 = x^3 + b。$$

令 $\beta \in \mathbb{F}_p$ 是一个阶为 3 的元素。则由

$$\phi: (x, y) \mapsto (\beta x, y), \quad \phi: \infty \mapsto \infty$$

所定义的映射 ϕ：$E \to E$ 是 $\mathbb{F}_p$ 上的椭圆曲线 E 的一个自同态。注意，$\phi(P)$ 仅用一次乘就可以计算。ϕ 的特征多项式是 $X^2 + X + 1$。

注 3.72 （自同态的整数表示）设 E 是定义在有限域 $\mathbb{F}_q$ 上的椭圆曲线，还设 $\#E(\mathbb{F}_q)$ 能被一个素数 n 除尽，并且 n^2 除不尽 $\#E(\mathbb{F}_q)$。则 $E(\mathbb{F}_q)$ 恰好包含一个 n 阶子群，设这一 n 阶子群为 $\langle P \rangle$，其中 $P \in E(\mathbb{F}_q)$ 且阶为 n。若 ϕ 是定义在 $\mathbb{F}_q$ 上的椭圆曲线 E 的一个自同态，则 $\phi(P) \in E(\mathbb{F}_q)$，因此 $\phi(P) \in \langle P \rangle$。假设 $\phi(P) \neq \infty$，我们记

$$\phi(P) = \lambda P，\text{对于某些} \lambda \in [1, n-1]。$$

实际上，λ 是 ϕ 的特征方程的 mod n 根。

例 3.73 （P-160 椭圆曲线）考虑定义在 160 位的素域 $\mathbb{F}_p$ 上的椭圆曲线

$$E: y^2 = x^3 + 3,$$

其中 $p = 2^{160} - 229233$

$= 1461501637330902918203684832716283019655932313743$。

因为 $p \equiv 1 \pmod 3$，所以曲线属于例 3.71(iv)中描述的类型。E 的 $\mathbb{F}_p$ 有理数点群的阶为素数

$$\#E(\mathbb{F}_p) = n = 1461501637330902918203687013445034429194588307251。$$

$\mathbb{F}_p$ 中的一个 3 阶元素是

$$\beta = 771473166210819779552257112796337671037538143582,$$

因此由 $\phi: \infty \mapsto \infty$ 且 $\phi: (x, y) \mapsto (\beta x, y)$ 所定义的映射 $\phi: E \to E$ 是定义在 $\mathbb{F}_p$ 上的椭圆曲线 E 的一个自同态。方程 $\lambda^2 + \lambda + 1 \equiv 0 \pmod n$ 的根

$$\lambda = 903860042511079968555273866340564498116022318806$$

具有如下特性：对于所有的 $P \in E(\mathbb{F}_p)$，$\phi(P) = \lambda P$。

加速点乘

计算 kP 的方法如下，其中 $k \in [0, n-1]$。首先记

$$k = k_1 + k_2\lambda \bmod n, \tag{3.36}$$

其中 k_1 和 k_2 的位数近似为 k 的位数的一半。这种表示式称为 k 的平衡长度 2 表示式。因为

$$\begin{aligned} kP &= k_1P + k_2\lambda P \\ &= k_1P + k_2\phi(P), \end{aligned} \tag{3.37}$$

所以 kP 可以如下计算：首先计算 $\phi(P)$，然后利用同时多点乘算法（算法 3.48）或交错（算法 3.51）来赋值式(3.37)。因为 k_1 和 k_2 的位数近似为 k 的位数一半，所以有一半的倍点运算被省略了。这种方法使得分解式(3.36)和 $\phi(P)$ 都能有效地计算。

分解一个乘数

现在我们讲述一种获得乘数 k 的平衡长度 2 表示式的方法。对于一个向量 $v = (a, b) \in \mathbb{Z} \times \mathbb{Z}$，定义

$$f(v) = a + b\lambda \bmod n。$$

首先，寻找两个向量 $v_1 = (a_1, b_1)$ 和 $v_2 = (a_2, b_2) \in \mathbb{Z} \times \mathbb{Z}$，使得

1. v_1 和 v_2 在 $\mathbb{R}$ 上线性无关。
2. $f(v_1) = f(v_2) = 0$。
3. v_1 和 v_2 有小的 Euclidean 范数（即 $\|v_1\| = \sqrt{a_1^2 + b_1^2} \approx \sqrt{n}$，$v_2$ 类似）。

然后，把 $(k, 0)$ 视为 $\mathbb{Q} \times \mathbb{Q}$ 中的一个向量，我们根据线性代数的基本知识得到

$$(k, 0) = \gamma_1 v_1 + \gamma_2 v_2，\text{其中} \gamma_1, \gamma_2 \in \mathbb{Q}。$$

若我们令 $c_1 = \lfloor \gamma_1 \rceil$ 和 $c_2 = \lfloor \gamma_2 \rceil$，其中 $\lfloor x \rceil$ 表示最靠近 x 的整数，则 $v = c_1 v_1 + c_2 v_2$ 是一个使得 $f(v) = 0$ 的最靠近 $(k, 0)$ 的整数值向量。因此，向量 $u = (k, 0) - v$ 具有小的范数，并且满足 $f(u) = k$。由此得出，u 的元素 k_1 和 k_2 的绝对值小，并且满足 $k_1 + k_2\lambda \equiv k \pmod n$。

通过对 n 和 λ 应用扩展的 Euclidean 算法（算法 2.19），可以找到满足 $f(v_1) = f(v_2) = 0$ 的独立的短向量 v_1 和 v_2。算法产生一个方程序列 $s_i n + t_i \lambda = r_i$，其中 $s_0 = 1$，$t_0 = 0$，$r_0 = n$，$s_1 = 0$，$t_1 = 1$，$r_1 = \lambda$。进而，容易看出余数 r_i 是严格递减的，而且非负。即对于 $i \geqslant 0$ 有 $|t_i| < |t_{i+1}|$，且对于 $i \geqslant 1$ 有 $|s_i| < |s_{i+1}|$ 和 $r_{i-1}|t_i| + r_i|t_{i-1}| = n$。现在，令 l 是使 $r_l \geqslant \sqrt{n}$ 的最大下标。则容易验证 $v_1 = (r_{l+1}, -t_{l+1})$ 满足 $f(v_1) = 0$，且 $\|v_1\| \leqslant \sqrt{2n}$ 和 $v_2 = (r_l, -t_l)$（还有 $v_2 = (r_{l+2}, -t_{l+2})$）是与 v_1 线性无关的，$f(v_2) = 0$。受此启发，我们期望 v_2 有小的范数。因此，v_1 和 v_2 满足上述条件 1~3。对于这样选择的 v_1 和 v_2，我们有 $\gamma_1 = b_2 k/n$ 和

$\gamma_2 = -b_1 k/n$。上述分解 k 的方法总结于算法 3.74 中。

算法 3.74 乘数的平衡长度 2 表示式

输入：整数 $n, \lambda, k \in [0, n-1]$。

输出：整数 k_1, k_2，$k = k_1 + k_2\lambda \bmod n$ 且 $|k_1|, |k_2| \approx \sqrt{n}$。

1. 对输入 n 和 λ，应用扩展的 Euclidean 算法（算法 2.19）。算法产生一个方程序列 $s_i n + t_i \lambda = r_i$，其中 $s_0 = 1, t_0 = 0$，$r_0 = n, s_1 = 0, t_1 = 1, r_1 = \lambda$，并且余数 r_i 是非负且严格递减的。令 l 是使 $r_l \geqslant \sqrt{n}$ 的最大下标。
2. $(a_1, b_1) \leftarrow (r_{l+1}, -t_{l+1})$。
3. 若 $(r_l^2 + t_l^2) \leqslant (r_{l+2}^2 + t_{l+2}^2)$，则 $(a_2, b_2) \leftarrow (r_l, -t_l)$；否则，$(a_2, b_2) \leftarrow (r_{l+2}, -t_{l+2})$。
4. 计算 $c_1 = \lfloor b_2 k/n \rceil$ 和 $c_2 = \lfloor -b_1 k/n \rceil$。
5. 计算 $k_1 = k - c_1 a_1 - c_2 a_2$ 和 $k_2 = -c_1 b_1 - c_2 b_2$。
6. 返回 (k_1, k_2)。

例 3.75 （乘数 k 的平衡长度 2 表示式）考虑例 3.73 中所定义的椭圆曲线 P-160。使用算法 3.74 中的符号，我们有

$$\begin{aligned}
(r_l, t_l) &= (2180728751409538655993509, -1860295391676851993530 61), \\
(r_{l+1}, t_{l+1}) &= (7889194301924079517821 90, 6028898910247227524291 29), \\
(r_{l+2}, t_{l+2}) &= (6028898910247227524291 29, -1391809321217130704211319), \\
(a_1, b_1) &= (7889194301924079517821 90, -6028898910247227524291 29), \\
(a_2, b_2) &= (6028898910247227524291 29, 1391809321217130704211319)。
\end{aligned}$$

现在，令

$$k = 965486288327218559097909069724275579360008398257。$$

我们得到

$$c_1 = 919446671339517233512759, \quad c_2 = 398276613783683332374156$$

和

$$k_1 = -98093723971803846754077, \quad k_2 = 381880690058693066485147。$$

例 3.76 （对于特殊参数的平衡表示式）巧妙选择椭圆曲线，可以使得用比算法 3.74 所需小得多的代价就能得到 k_1 和 k_2。例如，考虑定义在 $\mathbb{F}_p$ 上的曲线

$$E: y^2 = x^3 - 2,$$

其中$p = 2^{390} + 3$是素数，并且满足$p \equiv 1 \pmod 3$，参见例 3.71(iv)。E 的$\mathbb{F}_p$上的有理数点群的阶为

$$\#E(\mathbb{F}_p) = 2^{390} - 2^{195} + 7 = 63n,$$

其中 n 是一个素数。若

$$\lambda = \frac{2^{195} - 2}{3} \quad 和 \quad \beta = 2^{389} + 2^{194} + 1,$$

则β是$\mathbb{F}_p$上的 3 阶元素，λ满足$\lambda^2 + \lambda + 1 \equiv 0 \pmod n$，并且在$E(\mathbb{F}_p)$的 n 阶子群中对于所有的(x, y)都有$\lambda(x, y) = (\beta x, y)$。

现在假设$P = (x, y)$在$E(\mathbb{F}_p)$的 n 阶子群中，并且$k \in [0, n-1]$为乘数。为了获得 k 的平衡长度 2 表示式，记$k = 2^{195}k_2' + k_1'$，$k_1' < 2^{195}$。则

$$\begin{aligned} kP &= (2^{195}k_2' + k_1')P = ((3\lambda + 2)k_2' + k_1')P = \underbrace{(2k_2' + k_1')}_{k_1} P + \underbrace{3k_2'}_{k_2} \lambda P \\ &= k_1(x, y) + k_2(\beta x, y)。 \end{aligned}$$

这种方法把一个近似 384 位的小于 n 的乘数 k，转换为两个长度约为 k 的一半的数 k_1 和 k_2。最后，注意计算 $\beta x = (2^{389} + 2^{194} + 1)x$的开销小于域的乘法。

点乘算法

给定一个定义在$\mathbb{F}_q$上的具有合适的自同态ϕ的椭圆曲线 E，算法 3.77 计算 kP，采用将 k 分解为$k = k_1 + k_2\lambda$和交错$k_1 P + k_2\phi(P)$的技术。期望的运行时间近似为

$$\left[|\{j : w_j > 2\}| D + \sum_{j=1}^{2} (2^{w_j - 2} - 1)A + C_k + C_\phi \right] + \left[D + \sum_{j=1}^{2} \frac{1}{w_j + 1} A \right] \frac{t}{2}, \qquad (3.38)$$

其中 t 是 n 的长度，k_j被写成宽度w_j的 NAF，C_k表示分解 k 的开销，C_ϕ表示寻找$\phi(P)$的开销。需要存储$2^{w_1 - 2} + 2^{w_2 - 2}$个点。

因为v_1和v_2与 k 独立，所以可以预计算b_1/n 和 $-b_2/n$，以便在算法 3.74 的步骤 4 中使用。在这种情况下，算法 3.74 中只有步骤 4~步骤 6 必须完成，因此在全部点乘中开销C_k是不大的。

算法 3.77　具有有效可计算自同态的点乘

输入：正整数$k \in [1, n-1]$, $P \in E(\mathbb{F}_q)$，窗口宽度 w_1和w_2，以及λ。

输出：kP。

1. 利用算法 3.74 寻找k_1和k_2，使得$k = k_1 + k_2\lambda \mod n$。
2. 计算$P_2 = \phi(P)$并令$P_1 = P$。
3. 对于$j = 1,2$，利用算法 3.30 计算$\mathrm{NAF}_{w_j}(|k_j|) = \sum_{i=0}^{l_j-1} k_{j,i}2^i$。
4. 令$l = \max\{l_1, l_2\}$，并且对于$l_j \leqslant i < l$，$1 \leqslant j \leqslant 2$定义$k_{j,i} = 0$。
5. 若$k_j < 0$，则$k_{j,i} \leftarrow -k_{j,i}$，$0 \leqslant i < l_j$，$1 \leqslant j \leqslant 2$。
6. 对于$i \in \{1,3,\ldots,2^{w_j-1}-1\}$和$1 \leqslant j \leqslant 2$，计算$iP_j$。
7. $Q \leftarrow \infty$。
8. 对于i从$l-1$到 0，重复执行
 8.1 $Q \leftarrow 2Q$。
 8.2 对于j从 1 到 2，重复执行
 若$k_{j,i} \neq 0$，则
 若$k_{j,i} > 0$，则$Q \leftarrow Q + k_{j,i}P_j$。
 否则，$Q \leftarrow Q - |k_{j,i}|P_j$。
9. 返回(Q)。

3.6 用折半进行点乘

从用快速运算代替倍点运算的意义上看，基于点折半的点乘方法享用了 Koblitz 曲线（见 3.4 节）的τ-adic方法的策略。虽然折半方法可用于较广泛的曲线类，但是与 3.5 节的有效可计算自同态一样，对运算性能的改善不像 Koblitz 曲线那样显著。

点的折半方法分别由 E. Knudsen 和 R. Schroeppel 独立提出。我们将把注意力集中到由下式定义的二进制域$\mathbb{F}_{2^m}$上的椭圆曲线E上：

$$y^2 + xy = x^3 + ax^2 + b,$$

其中$a, b \in \mathbb{F}_{2^m}$，$b \neq 0$。为了叙述简单，我们假设$\mathrm{Tr}(a) = 1$（参见定理 3.18）[①]。我们进一步假设$m$是素数并且模多项式是一个三项式或五项式。由 NIST 建议的作为 FIPS 186-2 标准的 5 个二进制域的随机曲线满足这些性质（见 A.2.2 节）。

令$P = (x, y)$是曲线E上的一个点，$P \neq -P$。根据 3.1.2 节的内容，可知$Q = 2P = (u, v)$

① 本节给出的算法能够改进二进制域上$\mathrm{Tr}(a) = 0$的曲线；然而，它们比$\mathrm{Tr}(a) = 1$的情况复杂。

的仿射坐标可以如下计算：

$$\lambda = x + y/x \tag{3.39}$$

$$u = \lambda^2 + \lambda + a \tag{3.40}$$

$$v = x^2 + u(\lambda + 1)。 \tag{3.41}$$

仿射倍点需要一次域乘法和一次域除法。对于投影坐标和$a \in \{0, 1\}$，倍点能用四次域乘法完成。点的折半是指如下操作：给定$Q = (u, v)$，计算 $P = (x, y)$，使得 $Q = 2P$。因为点的折半是倍点的逆操作，折半的基本思想是从式(3.40)中解出 λ，从式(3.41)中解出 x，最后求出式(3.39)中的 y。

当 $\mathcal{G}$ 是 E 的一个 n 阶（n 为奇数）子群时，倍点和点的折半是 $\mathcal{G}$ 的自同构。所以，给定一个点$Q \in \mathcal{G}$，总能找到惟一的一个点 $P \in \mathcal{G}$，使得 $Q = 2P$。3.6.1 节和 3.6.2 节论述了在 $\mathcal{G}$ 中进行点折半的有效算法。在 3.6.3 节中，点的折半被用来获得一种在二进制域椭圆曲线密码体制中高效的折半加（halve-and-add）点乘方法。

3.6.1　点的折半

注意，在导出有效的点的折半算法中，迹发挥了重要的作用。

定义 3.78　域$\mathbb{F}_{2^m}$上的迹函数是如下定义一个函数：

$$\mathrm{Tr} : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m},\ \mathrm{Tr}(c) = c + c^2 + c^{2^2} + \cdots + c^{2^{m-1}}。$$

引理 3.79　（迹函数的性质）令$c, d \in \mathbb{F}_{2^m}$。

(i) $\mathrm{Tr}(c) = \mathrm{Tr}(c^2) = \mathrm{Tr}(c)^2$；特别地，$\mathrm{Tr}(c) \in \{0, 1\}$。

(ii) 迹是线性的；即$\mathrm{Tr}(c + d) = \mathrm{Tr}(c) + \mathrm{Tr}(d)$。

(iii) 若$(u, v) \in \mathcal{G}$，则$\mathrm{Tr}(u) = \mathrm{Tr}(a)$。

可从式(3.40)导出性质(iii)，这是因为

$$\mathrm{Tr}(u) = \mathrm{Tr}(\lambda^2 + \lambda + a) = \mathrm{Tr}(\lambda)^2 + \mathrm{Tr}(\lambda) + \mathrm{Tr}(a) = \mathrm{Tr}(a)。$$

给定$Q = (u, v) \in \mathcal{G}$，点的折半就是寻找惟一的一个点 $P = (x, y) \in \mathcal{G}$ suc，使得 $Q = 2P$。折半的第一步是通过对$\widehat{\lambda}$求解方程(3.42)寻找$\lambda = x + y/x$。

$$\widehat{\lambda}^2 + \widehat{\lambda} = u + a \tag{3.42}$$

求解方程(3.42)的一个有效算法在 3.6.2 节中给出。令$\widehat{\lambda}$表示通过这一算法求得的方程(3.42)的解。容易验证，$\lambda \in \{\widehat{\lambda}, \widehat{\lambda} + 1\}$。若$\mathrm{Tr}(a) = 1$，则下列结果能够用于识别$\lambda$。

定理 3.80 令 $P=(x,y), Q=(u,v)\in\mathcal{G}$，而 $Q=2P$，$\lambda=x+y/x$。再令 $\widehat{\lambda}$ 是方程(3.42)的一个解，并且 $t=v+u\widehat{\lambda}$。假设 $\mathrm{Tr}(a)=1$，则 $\widehat{\lambda}=\lambda$，当且仅当 $\mathrm{Tr}(t)=0$。

证明：根据式(3.41)有 $x^2=v+u(\lambda+1)$。由引理 3.79(iii)，我们得到 $\mathrm{Tr}(x)=\mathrm{Tr}(a)$，因为 $P=(x,y)\in\mathcal{G}$。所以

$$\mathrm{Tr}(v+u(\lambda+1))=\mathrm{Tr}(x^2)=\mathrm{Tr}(x)=\mathrm{Tr}(a)=1。$$

因此，若 $\widehat{\lambda}=\lambda+1$，则 $\mathrm{Tr}(t)=\mathrm{Tr}(v+u(\lambda+1))=1$ 是需要的。否则，我们必须有 $\widehat{\lambda}=\lambda$，使得 $\mathrm{Tr}(t)=\mathrm{Tr}(v+u\lambda)=\mathrm{Tr}(v+u((\lambda+1)+1))$。又因为迹是线性的，从而有

$$\mathrm{Tr}(v+u((\lambda+1)+1))=\mathrm{Tr}(v+u(\lambda+1))+\mathrm{Tr}(u)=1+\mathrm{Tr}(u)=0。$$

所以我们得到结论：$\widehat{\lambda}=\lambda$，当且仅当 $\mathrm{Tr}(t)=0$。

定理 3.80 隐含着一种在 $\mathrm{Tr}(a)=1$ 的条件下识别 λ 的简单算法。我们可以求解方程 $x^2=v+u(\lambda+1)$，解出惟一的根 x。3.6.2 节中给出了在域 $\mathbb{F}_{2^m}$ 上寻找迹和平方根的有效算法。最后，如果需要，用一次域乘法就可得出 $y=\lambda x+x^2$。

令 (u,λ_Q) 是点 $Q=(u,v)$ 的 λ 表示，其中，

$$\lambda_Q=u+\frac{v}{u}。$$

以给定的 Q 的 λ 表示为输入进行点的折半，我们可以根据定理 3.80 计算 t，而不必变换成仿射坐标，这是因为

$$t=v+u\widehat{\lambda}=u\left(u+u+\frac{v}{u}\right)+u\widehat{\lambda}=u(u+\lambda_Q+\widehat{\lambda})。$$

在点乘中，可以直接对点的 λ 表示完成重复折半，仅当需要点加时再变换成仿射坐标。

算法 3.81 点的折半

输入：点 $Q\in\mathcal{G}$ 的 λ 表示 (u,λ_Q) 或仿射表示 (u,v)。

输出：点 $P=(x,y)\in\mathcal{G}$ 的 λ 表示 (x,λ_P)，其中 $Q=2P$。

1. 求解 $\widehat{\lambda}^2+\widehat{\lambda}=u+a$，获得 $\widehat{\lambda}$。
2. 若输入是 λ 表示，则计算 $t=u(u+\lambda_Q+\widehat{\lambda})$。
 否则，计算 $t=v+u\widehat{\lambda}$。
3. 若 $\mathrm{Tr}(t)=0$，则 $\lambda_P\leftarrow\widehat{\lambda}, x\leftarrow\sqrt{t+u}$。
 否则，$\lambda_P\leftarrow\widehat{\lambda}+1, x\leftarrow\sqrt{t}$。
4. 返回 (x,λ_P)。

3.6.2　点折半的有效实现

点的折半需要一次域乘法和三个主要步骤：(i) 计算 t 的迹；(ii) 解二次方程(3.42)；(iii) 计算一个平方根。对于正规基表示，域的元素用基$\{\beta, \beta^2, \ldots, \beta^{2^{m-1}}\}$中的项的组合来表示。元素$c=\sum c_i\beta^{2^i}=(c_0, c_1, \ldots, c_{m-1})$的迹由$\mathrm{Tr}(c)=\sum c_i$给出。平方根的计算用左循环移位来实现：$\sqrt{c}=(c_1, \ldots, c_{m-1}, c_0)$。而平方则用右循环移位来实现，并且$x^2+x=c$可以按位的形式求解。这些运算相对于域的乘法是比较容易的。然而，与多项式基表示相比，用软件实现正规基表示的域乘法是相当慢的。每次对点折半时进行多项式和正规基的变换未必是一种可取的方法，即使是有大量的存储器可用。根据此原因，我们把讨论限制在多项式基表示中。

迹的计算

令$c=\sum_{i=0}^{m-1}c_iz^i\in\mathbb{F}_{2^m}$，其中$c_i\in\{0,1\}$，将 c 表示为向量$c=(c_{m-1}, \ldots, c_0)$。计算$\mathrm{Tr}(c)$的一种基本方法是利用迹的定义。它需要 $m-1$ 次域的平方和 $m-1$ 次域的加法。一种更有效的计算方法是利用迹的线性特性：

$$\mathrm{Tr}(c)=\mathrm{Tr}\left(\sum_{i=0}^{m-1}c_iz^i\right)=\sum_{i=0}^{m-1}c_i\mathrm{Tr}(z^i)。$$

其中的$\mathrm{Tr}(z^i)$可以预先计算，以使域元素的迹的计算能有效进行。若对于大多数的 i 有$\mathrm{Tr}(z^i)=0$，则计算会更有效。

例 3.82　（计算域$\mathbb{F}_{2^{163}}$中元素的迹）考虑约减多项式为$f(z)=z^{163}+z^7+z^6+z^3+1$的域$\mathbb{F}_{2^{163}}$。计算表明，$\mathrm{Tr}(z^i)=1$，当且仅当$i\in\{0, 157\}$。作为一个例子，$\mathrm{Tr}(z^{160}+z^{46})=0$，$\mathrm{Tr}(z^{157}+z^{46})=1$，并且$\mathrm{Tr}(z^{157}+z^{46}+1)=0$。

求解二次方程

点折半的第一步是寻找域$\mathbb{F}_{2^m}$上的二次方程 $x^2+x=c$的解 x，这一步的执行时间对于有效地进行点折半是至关重要的。

定义 3.83　令 m 是一个奇整数，半迹函数$H:\mathbb{F}_{2^m}\to\mathbb{F}_{2^m}$定义为

$$H(c)=\sum_{i=0}^{(m-1)/2}c^{2^{2i}}。$$

引理 3.84 （半迹函数的性质）令 m 是一个奇整数。

(i) $H(c+d)=H(c)+H(d)$，对于所有的$c,d\in\mathbb{F}_{2^m}$。

(ii) $H(c)$是方程$x^2+x=c+\mathrm{Tr}(c)$的一个解。

(iii) $H(c)=H(c^2)+c+\mathrm{Tr}(c)$，对于所有的$c\in\mathbb{F}_{2^m}$。

令$c=\sum_{i=0}^{m-1}c_iz^i\in\mathbb{F}_{2^m}$，$\mathrm{Tr}(c)=0$。特别地，$H(c)$是方程$x^2+x=c$的一个解。确定$H(c)$的一种简单方法是直接根据定义进行，这将需要$m-1$次平方和$(m-1)/2$次加法。若可以存储$\{H(z^i):0\leqslant i<m\}$，则可根据引理 3.84(i)计算获得

$$H(c)=H\left(\sum_{i=0}^{m-1}c_iz^i\right)=\sum_{i=0}^{m-1}c_iH(z^i)。$$

然而这需要存储 m 个域元素，相应的计算平均需要 $m/2$ 次域加法。

引理 3.84 能够大大减少所需要的存储量和求解二次方程所需要的时间。这一方法的基本策略是记$H(c)=H(c')+s$，其中c'比c有更少的非零系数。对于偶数的i，有

$$H(z^i)=H(z^{i/2})+z^{i/2}+\mathrm{Tr}(z^i)。$$

算法 3.85 建立在如下的考虑之上，即对于所有的偶数i，省去$H(z^i)$的存储。对于奇数i，进行预计算并建立有着$(m-1)/2$个域元素$H(z^i)$的表，则算法的步骤 4 约有$m/4$次域加法。涉及$\mathrm{Tr}(z^i)$和$H(1)$的项都被忽略，因为已经足够产生方程$x^2+x=c$的一个解$s\in\{H(c),H(c)+1\}$。

算法 3.85 解方程$x^2+x=c$（基本形式）

输入：$c=\sum_{i=0}^{m-1}c_iz^i\in\mathbb{F}_{2^m}$，其中$m$是奇数且$\mathrm{Tr}(c)=0$。

输出：方程$x^2+x=c$的解s。

1. 对于奇数的i，$1\leqslant i\leqslant m-2$，预计算$H(z^i)$。
2. $s\leftarrow 0$。
3. 对于i从$(m-1)/2$到1，重复执行
 3.1 若$c_{2i}=1$，则$c\leftarrow c+z^i$，$s\leftarrow s+z^i$。
4. $s\leftarrow s+\sum_{i=1}^{(m-1)/2}c_{2i-1}H(z^{2i-1})$。
5. 返回(s)。

应用引理 3.84 并结合约减多项式$f(z)$还可以对算法进行改进。令i是奇数，j和s按下

式确定：

$$m \leqslant 2^j i = m + s < 2m。$$

其基本思想是连续 j 次利用引理 3.84 (iii)，获得

$$H(z^i) = H(z^{2^j i}) + z^{2^{j-1} i} + \cdots + z^{4i} + z^{2i} + z^i + j\mathrm{Tr}(z^i)。 \qquad (3.43)$$

令 $f(z) = z^m + r(z)$，其中 $r(z) = z^{b_\ell} + \cdots + z^{b_1} + 1$ 且 $0 < b_1 < \cdots < b_\ell < m$。则

$$H(z^{2^j i}) = H(z^s r(z)) = H(z^{s+b_\ell}) + H(z^{s+b_{\ell-1}}) + \cdots + H(z^{s+b_1}) + H(z^s)。$$

这样，原来存储 $H(z^i)$ 现在可以换成存储 $H(z^{s+e})$，对于 $e \in \{0, b_1, \ldots, b_\ell\}$（其中一些还可以进一步节省）。所节省的存储量取决于元素 $H(z^i)$ 之间的独立性。

若 r 的次数小于 $m/2$，则可在算法 3.85 中直接省掉用于存储 $H(z^i)$ 的某些元素的存储器。对于 $m/2 < i < m - \deg r$，

$$\begin{aligned} H(z^i) &= H(z^{2i}) + z^i + \mathrm{Tr}(z^i) \\ &= H(r(z) z^{2i-m}) + z^i + \mathrm{Tr}(z^i) \\ &= H(z^{2i-m+b_\ell} + \cdots + z^{2i-m+b_1} + z^{2i-m}) + z^i + \mathrm{Tr}(z^i)。 \end{aligned}$$

因为 $2i - m + \deg r < i$，对于奇数 i，$m/2 < i < m - \deg r$，约减可以用于节省存储 $H(z^i)$ 的存储器。若 $\deg r$ 很小，则算法 3.86 计算近似需要存储 $m/4$ 个元素。

算法 3.86　解方程 $x^2 + x = c$

输入：$c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$，其中 m 是奇数且 $\mathrm{Tr}(c) = 0$，约减多项式 $f(z) = z^m + r(z)$。

输出：方程 $x^2 + x = c$ 的解 s。

1. 对于 $i \in I_0 \cup I_1$，预计算 $H(z^i)$。其中 I_0 和 I_1 分别由集合 $[1, (m-1)/2]$ 和 $[m - \deg r, m-2]$ 的奇整数组成。
2. $s \leftarrow 0$。
3. 对于每个奇数 $i \in ((m-1)/2, m - \deg r)$，按递减顺序处理，执行
 3.1　若 $c_i = 1$，则 $c \leftarrow c + z^{2i-m+b_\ell} + \cdots + z^{2i-m}$，$s \leftarrow s + z^i$。
4. 对于 i 从 $(m-1)/2$ 到 1，重复执行
 4.1　若 $c_{2i} = 1$，则 $c \leftarrow c + z^i$，$s \leftarrow s + z^i$。
5. $s \leftarrow s + \sum_{i \in I_0 \cup I_1} c_i H(z^i)$。
6. 返回(s)。

还可以进一步减少求解二次方程所需要的时间，因为约减每个$H(z^i)$的开销可能小于把预计算的$H(z^i)$值加到累加器的开销。在步骤4省掉偶数项，可以使执行更有效。奇数项的处理（如步骤3）是必须包括的，但若只有少量的字需要更新，则与域加法相比并不是很费时。

例3.87 （域$\mathbb{F}_{2^{163}}$的算法3.86）考虑约减多项式$f(z)=z^{163}+z^7+z^6+z^3+1$的域$\mathbb{F}_{2^{163}}$。算法3.86的步骤3从$i=155$开始。由引理3.84，我们有

$$\begin{aligned}H(z^{155}) &= H(z^{310})+z^{155}+\mathrm{Tr}(z^{155})\\ &= H(z^{147}z^{163})+z^{155}\\ &= H(z^{147}(z^7+z^6+z^3+1))+z^{155}。\end{aligned}$$

若$c_{155}=1$，则$z^{154}+z^{153}+z^{150}+z^{147}$加$c$，且$z^{155}$加$s$。在这种情况下，对于$i\in\{83,85,\ldots,155\}$，即在范围$((m-1)/2, m-\deg r)$中的奇整数，$H(z^i)$的存储可以省去。

算法3.86使用了44个域元素的预计算。然而这仅是基本算法所需要的一半左右，并不是最小值。例如，$H(z^{51})$的存储可以省去，因为

$$\begin{aligned}H(z^{51}) &= H(z^{102})+z^{51}+\mathrm{Tr}(z^{51})\\ &= H(z^{204})+z^{102}+z^{51}+\mathrm{Tr}(z^{102})+\mathrm{Tr}(z^{51})\\ &= H(z^{163}z^{41})+z^{102}+z^{51}\\ &= H(z^{48}+z^{47}+z^{44}+z^{41})+z^{102}+z^{51},\end{aligned}$$

这相当于$j=2$时的式(3.43)。同理可省去$H(z^i)$的存储，$i\in\{51,49,\ldots,41\}$。类似地，若对于$i=21$和$j=3$应用式(3.43)，则有

$$H(z^{21})=H(z^{12}+z^{11}+z^8+z^5)+z^{84}+z^{42}+z^{21}。$$

注意，奇指数11和5小于21，因此存储$H(z^{21})$的存储器可以省去。

总之，对应于$j\in\{1,2,3\}$应用式(3.43)可省去i为奇数值时的存储器，其中$i\in\{21,41,\ldots,51,83,\ldots,155\}$。求解二次方程的相应算法需要预计算37个域元素。进一步的节省是可能的，但这相当复杂，因为关于$H(z^i)$的计算必须涉及$H(z^j)$，$j>i$。例如，

$$H(z^{23})=H(z^{28}+z^{27}+z^{24}+z^{21})+z^{92}+z^{46}+z^{23}$$

对于$H(z^{23})$的存储可以改变为存储$H(z^{27})$。这一方法把预计算的量减少到30个域元素，大大小于算法3.86所需要的44个。实际上，应用

$$z^n = z^{157+n} + z^{n+1} + z^{n-3} + z^{n-6}$$

并结合以前的节省存储器的技术，可以把存储量减少到 21 个域元素 $H(z^i)$，其中 $i \in \{157, 73, 69, 65, 61, 57, 53, 39, 37, 33, 29, 27, 17, 15, 13, 11, 9, 7, 5, 3, 1\}$。然而，与预计算 30 个元素时的开销相比，最后的约减在代码方面需要较高的开销。

实验上，求解二次方程的算法（使用 21 个或 30 个元素的预计算）所花费的时间大约是一个域乘法的 2/3。应当特别关注分支预测失误因素（见 5.1.4 节），因为这种算法会执行许多位测试。

在 $\mathbf{F}_{2^m}$ 上求平方根

计算 $\sqrt{c}$ 的基本方法建立在小 Fermat 定理 $c^{2^m} = c$ 的基础之上，其中 $c \in \mathbb{F}_{2^m}$。因此求 $\sqrt{c}$ 可以按 $\sqrt{c} = c^{2^{m-1}}$ 来计算，这需要计算 $m-1$ 次平方。一种更有效的方法基于以下事实：$\sqrt{c}$ 能用元素 z 的平方根来表示。令 $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$，$c_i \in \{0, 1\}$。因为平方是域 $\mathbb{F}_{2^m}$ 上的线性运算，所以 c 的平方根可以写成

$$\sqrt{c} = \left(\sum_{i=0}^{m-1} c_i z^i\right)^{2^{m-1}} = \sum_{i=0}^{m-1} c_i (z^{2^{m-1}})^i 。$$

把 c 分为奇数幂和偶数幂，我们有

$$\begin{aligned}\sqrt{c} &= \sum_{i=0}^{(m-1)/2} c_{2i} (z^{2^{m-1}})^{2i} + \sum_{i=0}^{(m-3)/2} c_{2i+1} (z^{2^{m-1}})^{2i+1} \\ &= \sum_{i=0}^{(m-1)/2} c_{2i} z^i + \sum_{i=0}^{(m-3)/2} c_{2i+1} z^{2^{m-1}} z^i \\ &= \sum_{i \text{ even}} c_i z^{\frac{i}{2}} + \sqrt{z} \sum_{i \text{ odd}} c_i z^{\frac{i-1}{2}} 。\end{aligned}$$

这一表达式给出了一种计算 $\sqrt{c}$ 的有效方法：从 c（假设 m 是奇数）分出两个半长的向量 $c_{\text{even}} = (c_{m-1}, \ldots, c_4, c_2, c_0)$ 和 $c_{\text{odd}} = (c_{m-2}, \ldots, c_5, c_3, c_1)$，用预计算的 $\sqrt{z}$ 与一个长度为 $\lfloor m/2 \rfloor$ 的 c_{odd} 相乘，最后再加上 c_{even}。这一计算需要的时间近似为一次域乘法的一半。

在约减多项式 f 为三项式的情况下，$\sqrt{c}$ 的计算可进一步加速，根据是计算 $\sqrt{z}$ 的一种有效公式可直接从 f 导出。令 $f(z) = z^m + z^k + 1$ 是一个次数为 m 的既约三项式，其中 $m > 2$ 且为素数。

考虑 k 是奇数的情况。注意到 $1 \equiv z^m + z^k \pmod{f(z)}$。先用 z 乘再开平方可得

$$\sqrt{z} \equiv z^{\frac{m+1}{2}} + z^{\frac{k+1}{2}} (\mathrm{mod}\ f(z))。$$

因此，乘积 $\sqrt{z} \cdot c_{\mathrm{odd}}$ 需要两次左移操作和一次模约减运算。

现在假设 k 是偶数。考虑 $z^m \equiv z^k + 1\ (\mathrm{mod}\ f(z))$。先用 z^{m-1} 去除再开方可得

$$\sqrt{z} \equiv z^{-\frac{m-1}{2}} (z^{\frac{k}{2}} + 1)\ (\mathrm{mod}\ f(z))。$$

为了计算 $z^{-s}\ \mathrm{mod}\ f(z)$，其中 $s = \frac{m-1}{2}$，利用等式 $z^{-t} \equiv z^{k-t} + z^{m-t}\ (\mathrm{mod}\ f(z))$，$1 \leqslant t \leqslant k$，我们可以把 z^{-s} 表示为少数几个 z 的正数幂之和。因此，乘积 $\sqrt{z} \cdot c_{\mathrm{odd}}$ 可以用少量的左移操作和一次模约减运算来实现。

例 3.88 （$\mathbb{F}_{2^{409}}$ 上的平方根）NIST 建议的域 $\mathbb{F}_{2^{409}}$ 上的约减多项式是三项式 $f(z) = z^{409} + z^{87} + 1$。于是计算域 $\mathbb{F}_{2^{409}}$ 上的平方根 $\sqrt{c}$ 的一个新公式为

$$\sqrt{c} = c_{\mathrm{even}} + z^{205} \cdot c_{\mathrm{odd}} + z^{44} \cdot c_{\mathrm{odd}} \mod f(z)。$$

例 3.89 （$\mathbb{F}_{2^{233}}$ 上的平方根）NIST 建议的域 $\mathbb{F}_{2^{233}}$ 上的约减多项式是三项式 $f(z) = z^{233} + z^{74} + 1$。因为 $k = 74$ 为偶数，所以我们有 $\sqrt{z} = z^{-116} \cdot (z^{37} + 1)\ \mathrm{mod}\ f(z)$。注意，$z^{-74} \equiv 1 + z^{159}\ (\mathrm{mod}\ f(z))$ 并且 $z^{-42} \equiv z^{32} + z^{191}\ (\mathrm{mod}\ f(z))$。因此得到 $z^{-116} \equiv z^{32} + z^{117} + z^{191}\ (\mathrm{mod}\ f(z))$。从而计算域 $\mathbb{F}_{2^{233}}$ 上的平方根 $\sqrt{c}$ 的新公式为

$$\sqrt{c} = c_{\mathrm{even}} + (z^{32} + z^{117} + z^{191})(z^{37} + 1) \cdot c_{\mathrm{odd}} \mod f(z)。$$

与计算平方根的标准方法相比，这里所给出的方法节省了所需的存储空间，并且用一种快速运算代替了复杂的域乘法。根据经验，对于例 3.89，求平方根需要的时间近似为域乘法的 1/8。

3.6.3 点乘

3.3 节介绍的改进的折半加点乘方法用折半代替了大量的点乘。根据应用，采用基于折半的方法，可能需要对整数 $k = (k_{t-1}, \ldots, k_0)_2$ 进行变换。若 k' 定义为

$$k \equiv k'_{t-1}/2^{t-1} + \cdots + k'_2/2^2 + k'_1/2 + k'_0 \pmod{n},$$

则 $kP = \sum_{i=0}^{t-1} k'_i/2^i P$，即 $(k'_{t-1}, \ldots, k'_0)$ 用于基于折半的方法。这样就扩展了宽度 w 的 NAF 方法。

引理 3.90 令 $\sum_{i=0}^{t} k'_i 2^i$ 为 $2^{t-1}k \mod n$ 的 w-NAF 表示，则

$$k \equiv \sum_{i=0}^{t-1} \frac{k'_{t-1-i}}{2^i} + 2k'_t \pmod{n}。$$

证明：我们有 $2^{t-1}k \equiv \sum_{i=0}^{t} k'_i 2^i \pmod{n}$。因为 n 是素数，用 2^{t-1}除此等式可得

$$k \equiv \sum_{i=0}^{t} \frac{k'_i}{2^{t-1-i}} \equiv \sum_{i=0}^{t-1} \frac{k'_{t-1-i}}{2^i} + 2k'_t \pmod{n}。$$

算法 3.91 给出了一种从右向左的折半加方法，其输入为 $2^{t-1}k \bmod n$且用w-NAF表示。对点的折半是对输入 P 而不是对累加器进行。注意，系数k'_t在算法的步骤 2 分别处理，因为它与 k 中特定的 $2k'_t$项相对应。期望的运行时间近似为

$$（步骤 4 的开销）+ (t/(w+1) - 2^{w-2})A' + tH，\tag{3.44}$$

其中 H 表示点的折半，A'是当输入之一为λ表示时的点加的开销。若 Q_i采用投影坐标，则在算法的步骤 3.1 和步骤 3.2 的加法是按混合坐标进行的。算法的步骤 4 可以把 Q_i的坐标转换为仿射坐标来进行（开销为 $I + (5 \cdot 2^{w-2} - 3)M$，如果求逆也采用同样的方法），然后通过对奇乘数 i 的带符号数字表示法施加交错来求和。对于 $2 \leqslant w \leqslant 5$，算法的步骤 4 的开销近似为$w$–2 次倍点和分别为 0，2，6 或 16 次的点加[①]。

算法 3.91　折半加w-NAF（从右向左）点乘

输入：窗口宽度w，$\mathrm{NAF}_w(2^{t-1}k \bmod n) = \sum_{i=0}^{t} k'_i 2^i$，$P \in G$。

输出：kP（注意，$k = k'_0/2^{t-1} + \cdots + k'_{t-2}/2 + k'_{t-1} + 2k'_t \bmod n$）。

1. 对于 $i \in I = \{1, 3, \ldots, 2^{w-1} - 1\}$，令 $Q_i \leftarrow \infty$。
2. 若$k'_t = 1$，则 $Q_1 = 2P$。
3. 对于 i 从 $t-1$ 到 0，重复执行
 3.1 若 $k'_i > 0$，则$Q_{k'_i} \leftarrow Q_{k'_i} + P$。
 3.2 若 $k'_i < 0$，则 $Q_{-k'_i} \leftarrow Q_{-k'_i} - P$。
 3.3 $P \leftarrow P/2$。
4. $Q \leftarrow \sum_{i \in I} i Q_i$。
5. 返回(Q)。

① Knuth 建议对于 i 从 $2^{w-1}-3$到 1 计算 $Q_i \leftarrow Q_i + Q_{i+2}$，并且结果由 $Q_1 + 2\sum_{i \in I \setminus \{1\}} Q_i$给出。其计算开销与仿射坐标情况相当。

考虑 $w=2$ 的情况。算法 3.91 的期望运行时间近似为 $(1/3)tA'+tH$。若采用仿射坐标，则一次点折半的开销近似为 $2M$。而一次点加的开销为 $2M+V$，因为 P 的λ表示必须转化为仿射坐标，这要花费一次域乘法的开销。由此得出采用仿射坐标的域运算的开销近似为 $(8/3)tM+(1/3)tV$。然而，若 Q 按投影坐标存储，则一次点加需要 $9M$ 的开销。对于 $w=2$，采用混合坐标的算法 3.91 的域运算的量近似为 $5tM+(2M+I)$。

算法 3.92 是从左到右的方法。折半发生在累加器 Q，那时不能使用投影坐标。期望的运行时间近似为

$$(D+(2^{w-2}-1)A)+(t/(w+1)A'+tH)。 \tag{3.45}$$

算法 3.92 折半加 w-NAF（从左到右）点乘

输入：窗口宽度 w，$\mathrm{NAF}_w(2^{t-1}k \bmod n)=\sum_{i=0}^{t}k'_i2^i$，$P\in\mathcal{G}$。

输出：kP（注意，$k=k'_0/2^{t-1}+\cdots+k'_{t-2}/2+k'_{t-1}+2k'_t \mod n$）。

1. 对于 $i\in\{1,3,5,\ldots,2^{w-1}-1\}$，计算 $P_i=iP$。
2. $Q\leftarrow\infty$。
3. 对于 i 从 0 到 $t-1$，重复执行
 3.1 $Q\leftarrow Q/2$。
 3.2 若 $k'_i>0$，则 $Q\leftarrow Q+P_{k'_i}$。
 3.3 若 $k'_i<0$，则 $Q\leftarrow Q-P_{-k'_i}$。
4. 若 $k'_i=1$，则 $Q\leftarrow Q+2P$。
5. 返回(Q)。

分析

与基于倍点的方法相比，若 I/M 较小而且事先不知道 P 的情况下要计算 kP，则点折半方法看来更好。在应用中，对事先知道 P 而要计算 kP 和 $kP+lQ$ 也十分令人感兴趣，对此本节给出了比较结果。一个具体的应用实例是域 $\mathbb{F}_{2^{163}}$ 上的 NIST 随机曲线（见 A.2.2 节），虽然一般结论应用面更广泛。

例 3.93 （倍加方法与折半加方法之比较）表 3.11 给出了对于域 $\mathbb{F}_{2^{163}}$ 上的 NIST 随机曲线、倍加方法与折半加方法的计算量的比较。对于域运算，假设 $I/M=8$ 且域除法的开销为 $I+M$。

基本的 NAF 折半方法要优于 w-NAF 倍点方法。然而，折半方法要预计算 46 个域元素。相反，对于 $w=4$，算法 3.36（与 $w=5$ 的执行时间近似相等）仅需要额外存储 6 个域元素。

表 3.11　对于域 $\mathbb{F}_{2^{163}}$ 上的 NIST 随机曲线的点乘，点运算和域运算的计算量。在求解程序中，折半需要 30 个域元素的预计算，而求平方根又需要 16 个域元素的预计算。若一个输入用 λ 表示，则一个点加的开销为 $A' = A + M$。在域运算的计量中，假设除法 V 的开销为 $I + M$

方　法	存　储（域元素）	点运算	域运算（$H = 2M$，$I/M = 8$）	
			仿　射	投　影
NAF，倍点（算法 3.36）	0	$163D + 54A$	$217(M + V) = 2173$	$1089M + I = 1097$
NAF，折半（算法 3.91）	46	$163H + 54A'$	$435M + 54V = 924$	$817M + I = 825$
5-NAF，倍点（算法 3.36）	14	$[D+7A] + 163D + 27A$	$198(M + V) = 1982$	$979M + 8V + I = 959$
4-NAF，倍点（算法 3.91）	55	$[3D + 6A] + 163H + 30A'$	—	$671M + 2I = 687$
5-NAF，倍点（算法 3.92）	60	$[D + 7A] + 163H + 27A'$	$388M + 35V = 705$	—

从左向右的 w-NAF 折半方法需要累加器为仿射坐标，并且点加的开销为 $2M + V$（因为需要对λ表示进行变换）。对于足够大的 I/M，从右向左的算法将更好。例如，大约当 $I/M = 11$ 时，$w = 2$ 的算法 3.91 将优于算法 3.92。

对于点乘 kP，其中预先不知道 P，表 3.11 中的例子表明当 $I/M = 8$ 时，折半方法大约比基于倍点的同类方法节约 25%。

在所需存储方面的比较是不对等的，因为折半方法在求解程序和开平方根程序中共需要 46 个域元素的预计算。在折半方法开销容许的情况下，可以减少开平方根的存储量。主要的存储量（如 21~30 个元素）是求解所必需的。然而值得注意的是，求解程序和开平方根程序所需的存储是与域相关的。除了专门的折半程序外，对于基于倍点的方法的许多支持软件都是需要的，这会带来一些代码的扩充。

随机曲线与 Koblitz 曲线　Koblitz 曲线的 τ-adic 方法（见 3.4 节）与折半方法都采用了相同的策略，即用简单的运算代替倍点运算。对于 Koblitz 曲线的情况，代替是采用 Frobenius 映射 $\tau : (x, y) \mapsto (x^2, y^2)$，这是一个比域乘法简单的运算。用 τ-adic 方法进行 Koblitz 曲线的点乘快于基于折半的方法，当采用宽度 w 的 τ-adic NAF 方法（算法 3.70）时，kP 的开销近似为

$$\left(2^{w-2} - 1 + \frac{t}{w+1}\right)A + t \cdot (\tau \text{ 的开销})。$$

与表 3.11 比较，假设采用混合坐标、$w = 5$，并且域的平方的开销近似为 $M/6$。在这种情况下，运算量近似为 $379M$，大大小于折半方法所需的 $687M$。

已知点与未知点 在预先知道点 P（如 ECDSA 中的签名生成）和具有预计算所需的存储的情况下，折半方法失去了某些优势。对于我们的情况，即在存储空间适当的情况下，单表 comb 方法（算法 3.44）是其中最快的，并且能用来获得有意义的运算量的比较结果。使用基于倍点和折半的方法计算 kP 的运算量分别近似为

$$\frac{t}{w}\left(D+\frac{2^w-1}{2^w}A\right)\text{和}\frac{t}{w}\left(H+\frac{2^w-1}{2^w}A'\right)。$$

与随机点的情况相反，约有一半的运算是点加。注意，基于倍点的方法可以使用混合坐标（此种情况下 $D = 4M$，$A = 8M$，并且最后转化为仿射坐标）。而基于折半的方法必须采用仿射坐标（其中 $H = 2M$，$A' = V + 2M$）。若 $V = I + M$，则有实际意义的 t 和 w 的值给出了一个在 7 和 8 之间的门限 I/M 值，对于此种情况，基于倍点的方法预期是较好的（如对于 $w = 4$，$t = 163$，其门限值近似为 7.4）。

同时多点乘 在 ECDSA 签名验证过程中，一个比较费时的计算是计算 $kP + lQ$，其中只有 P 是预先知道的。若分别对宽度 w_1 和 w_2 实施交错（算法 3.51），则基于倍点的这种方法的期望运行时间近似为

$$[D+(2^{w_2-2}-1)A]+t\left[D+\left(\frac{1}{w_1+1}+\frac{1}{w_2+1}\right)A\right],$$

其中不包括 P 的预计算（基于折半方法的预期计算量可以用类似的公式来评估，然而更精确的评估必须区分发生连续加的情况，因为此时的开销是 $A' + V + M$ 而不是 $2A'$）。

对于足够大的 I/M，基于倍点的方法将更好；在例 3.93 中这种情况出现了，此时 $I/M = 11.7$。当 I/M 使得折半的方法更好时，就与随机点乘 kP 的情况差别不大了，因为相对于折半有大量的点加。注意，交错的方法并不能有效地转换为从右向左的算法（其中 $w_1 = w_2 = 2$），因为在每一步都需要进行两个点的折半或倍点运算。

3.7 点乘的开销

点乘算法的选择，由于要综合考虑平台特性、所用坐标、存储器大小、安全性要求（见 5.3 节）以及互操作性等方面的要求而十分复杂。本节给出以下三种情况下算法的基本比较：

NIST 推荐曲线 P-192（$p_{192} = 2^{192} - 2^{64} - 1$的素域$\mathbb{F}_{p_{192}}$），B-163 和 K-163（二进制域 $\mathbb{F}_{2^{163}} = \mathbb{F}_2[z]/(z^{163} + z^7 + z^6 + z^3 + 1)$ 上的随机曲线和 Koblitz 曲线）。一般做如下假设：素域中的求逆比乘法费时，具有预计算所需的适量存储空间，点运算的开销仅仅考虑域的乘法、平方和求逆运算。

椭圆曲线密码的执行时间基本上取决于点乘运算的时间。我们就以下三种情况给出点乘开销的评估：(i) 计算 kP，其中的预计算必须在线进行；(ii) 计算 kP，其中 P 预先知道并且预计算可以离线进行；(iii) 计算 $kP + lQ$，其中 P 的预计算可以离线进行。后两种情况是诸如 ECDSA 等协议所需要的，在 ECDSA 的签名生成过程中需要计算 kP，其中 P 是固定的。而在签名验证过程中需要计算 $kP + lQ$，其中 P 是固定的，而 Q 是事先未知的。

评估用椭圆曲线和域运算的术语给出（点加用 A 表示，倍点用 D 表示，域乘法用 M 表示，域上的求逆用 I 表示）。运算的量是针对算法所得到的近似表示；然而这些计算方法应考虑预计算和赋值阶段所采用的坐标以及各种细小的优化等因素。另一方面，关于标量的各种表示通常都假设为全长的，因此对某些量的估值偏高。然而，估值的方法对于所进行的比较是足够精确的。

评估 P-192

表3.12用椭圆曲线的运算和域运算的术语给出了对于P-192各种点乘方法的粗略评估，其中假设求逆运算大约相当于 80 个域乘法的运算。求逆运算的高开销促使人们采用投影坐标和同时求逆等技术。采用投影坐标的多数地方对 I/M 的精确值并不十分敏感，只要其值不是特别小。

对于点乘 kP，若预计算必须在线完成，则倍点的开销限制了基于基本 NAF 方法的窗口方法的改进。求逆与乘法相比占的比率太大会使得人们在窗口 NAF 方法的预计算中采用仿射坐标的 Chudnovsky 方法。固定基方法是非常快的（而且只需要很少的点存储空间），在那里预计算的开销节省了，并且在赋值阶段的倍点数也大大减少了。在计算 $kP + lQ$ 的多点乘方法中处理 Q 的开销，减弱了对于已知点的乘法减少倍点数量的技术的作用。另一方面，计算 $kP + lQ$ 的开销稍微高于未知点乘法的开销。

设 P 是固定的，采用交错技术计算 kP 意味着算法 3.51 采用如下输入：$v = 2$, $P_1 = P$, $P_2 = 2^d P$，半长标量k^1和k^2，k^1和k^2由 $k = 2^d k^2 + k^1$定义，其中$d = \lceil t/2 \rceil$。宽度 3 的 NAF 方法用于寻找每一个k^1和k^2。具有相同开销的另一种可选方法，采用改进的同时多点乘方法（算法 3.48）在每一步处理k^1和k^2的联合稀疏形（算法 3.50）的一列。这种改进的“同时 JSF”算法在表 3.12 的多点乘方法中被引用。

表 3.12 素数$p_{192}=2^{192}-2^{64}-1$的域$\mathrm{F}_{p_{192}}$上的 NIST 椭圆曲线的点乘开销的粗略评估。其中计算 kP 的未知点方法的开销中包括预计算的开销，而对于固定基方法不包括预计算的开销。计算 $kP+lQ$ 的多点乘方法中，仅 P 的预计算开销不计在内。假设求平方的开销 $S=0.85M$

方法	坐标	w	存储的点	椭圆曲线运算		域运算		
				A	D	M	I	合计[a]
未知点（kP，在线预计算）								
二进制	仿射	—	0	95	191	977	286	23 857
（算法 3.27）	雅可比仿射	—	0	95	191	2420	1	2500
二进制 NAF	仿射	—	0	63	191	886	254	21 206
（算法 3.31）	雅可比-仿射	—	0	63	191	2082	1	2162
窗口 NAF	雅可比-仿射	4	3	41	193	1840	4[b]	2160
（算法 3.36）	雅可比-Chudnovsky	5	7	38	192	1936	1	2016
固定基（kP，离线预计算）								
交错	雅可比-仿射	3, 3	3	47	95	1203	1	1283
（算法 3.51）								
窗口	Chudnovsky-仿射和	5	38	37[c]+30[d]	0	801	1	881
（算法 3.41）	雅可比-Chudnovsky							
窗口 NAF	Chudnovsky-仿射和	5	38	38[c]+20[d]	0	676	1	756
（算法 3.42）	雅可比-Chudnovsky							
Comb		5	30	37	38	675	1	755
（算法 3.44）	雅可比-仿射							
Comb2 表		4	29	44	23	638	1	718
（算法 3.45）	雅可比-仿射							
多点乘（$kP+lQ$）								
同时多点乘	雅可比-仿射和	2	10	91	192	2592	1	2672
（算法 3.48[f]）	雅可比-Chudnovsky							
同时 JSF	雅可比-仿射	—	2	97	191	2428	2[b]	2588
（算法 3.48 和 3.50）								
交错	雅可比-仿射和	6,5	22	32[e]+34[d]	192	2226	1	2306
（算法 3.51）	雅可比-Chudnovsky							

[a] 域乘法的总开销，设域的求逆的开销为 $I=80M$。

[b] 用于预计算的同时求逆。[c] $C+A\to C$，[d] $J+C\to J$，[e] $J+A\to J$。

[f] 采用滑动窗口。

评估 B-163 和 K-163

表 3.13 用椭圆曲线运算和域运算的术语给出了二进制域$\mathbb{F}_{2^{163}}$上的 NIST 随机曲线和 Koblitz 曲线（B-163 和 K-163）的各种点乘方法的粗略评估，其中$\mathbb{F}_{2^{163}} = \mathbb{F}_2[z]/(z^{163} + z^7 + z^6 + z^3 + 1)$。对于 TNAF 算法的评估是针对 K-163 的，而其他评估是针对 B-163 的。算法和坐标的选择是与求逆与乘法运算时间之比密切相关的，因为对于二进制域，这一比率远小于素域。进而，这种小的比率促使人们开发了仿射坐标的快速除法算法。

还给出了对于 $I/M = 5$ 和 $I/M = 8$ 情况下的评估，评估还假设域的除法 V 的开销近似为 $I + M$（即假设除法的开销近似与除法紧跟着乘法运算的开销相同），且域的平方运算相对于乘法运算是简单的。所做的假设和所考虑的情况是根据 5.1.5 节的通用硬件平台上的实验结果和实验数据来做出的。注意，若 $V/M \leqslant 7$，则仿射坐标在点加方面优于投影坐标。而投影坐标在倍点方面优于仿射坐标，除非 $V/M \leqslant 3$。

与对 P-192 的讨论类似，倍点开销的评估局限于针对 B-163 的基于基本 NAF 的各种窗口方法。然而，与 Koblitz 曲线的情况不同，这里倍点被简单的域平方所代替。然而，平方运算也不是完全无限制的，在对 TNAF 算法的评估中包括了域的平方，还假设域的平方采用 Frobenius 映射τ且其开销近似为 $S \approx M/7$。

基于点折半的方法（见 3.6 节）已经包括在未知点的计算情况中，并假设折半的开销近似为 $2M$。预计的运行时间明显优于 B-163 的情况，但是仍明显慢于特殊 Koblitz 曲线的 τ-adic 方法。然而注意，所列出的基于折半的方法所需的存储空间，没有考虑在求解和求平方根计算中所用的（固定的）域元素。类似地，还要注意 TNAF 程序需要τ-adic 方法的支持。

固定基方法相当快（而且只需要少数点的存储），其中预计算的开销未计在内，而且在赋值阶段倍点的个数也大大减少了。与 P-192 一样，在对 B-163 计算 $kP + lQ$ 的多点乘方法中处理 Q 点的开销，降低了在已知点乘法中减少倍点数量的技术的实用性。这不同于 Koblitz 曲线的情况，因为域的平方代替了多数倍点。

对 P-192 的讨论澄清了对固定点利用交错技术计算 kP 时的入口的意义。用于计算 $kP + lQ$ 的入口的 JSF 方法在本质上有相同的开销，并且很实用。通过调整处理 TNAF 的交错算法可以获得交错 TNAF 的入口。

表 3.13 对于域 $\mathbb{F}_{2^{163}}$ 上的 NIST 曲线的各种点乘开销的粗略评估，其中 $\mathbb{F}_{2^{163}}=\mathbb{F}_2[z]/(z^{163}+z^7+z^6+z^3+1)$。表中计算 kP 的未知点方法的开销包括预计算的开销，而对于固定基方法不包括预计算的开销。计算 $kP+lQ$ 的多点乘方法中，仅涉及 P 的预计算开销不计在内。预计算按仿射坐标进行

方法	坐标	w	存储的点	椭圆曲线运算		域运算			
				A	D	M	I	$I/M=5$	$I/M=8$
未知点（kP，在线预计算）									
二进制	仿射	0	0	81	162	486	243	1701	2430
（算法 3.27）	投影	0	0	81	162	1298	1	1303	1306
二进制 NAF	仿射	0	0	54	162	432	216	1512	2160
（算法 3.31）	投影	0	0	54	162	1082	1	1087	1090
窗口 NAF	仿射	4	3	35	163	396	198	1386	1980
（算法 3.36）	投影	4	3	3[b]+32	163	914	5	919	954
Montgomery	仿射	—	0	162[c]	162[d]	328	325	1953	2928
（算法 3.40）	投影	—	0	162[c]	162[d]	982	1	987	990
折半 w-NAF	仿射	5	7	7+27[e]	1+163[f]	423	35	598	705
（算法 3.91 和 3.92）	投影	4	3	6+30[e]	3+163[f]	671	2	681	687
TNAF	仿射	—	0	54	0[g]	154	54	424	586
（算法 3.66）	投影	—	0	54	0[g]	503	1	508	511
窗口 TNAF	仿射	5	7	34	0[g]	114	34	284	386
（算法 3.70）	投影	5	7	7[b]+27	0[g]	301	8	341	365
固定基（kP，离线预计算）									
交错	仿射	3,3	3	41	81	244	122	854	1220
（算法 3.51）	投影	3,3	3	41	81	654	1	659	662
窗口	仿射	5	32	61	0	122	61	427	610
（算法 3.41）	投影	5	32	31+30[h]	0	670	1	—	678
窗口 NAF	仿射	5	32	52	0	104	52	364	520
（算法 3.42）	投影	5	32	32+20[h]	0	538	1	—	546
Comb	仿射	5	30	31	32	126	63	441	630
（算法 3.44）	投影	5	30	31	32	378	1	383	386
窗口 TNAF	仿射	6	15	23	0[g]	92	23	207	276
（算法 3.70）	投影	6	15	23	0[g]	255	1	260	263
多点乘（$kP+lQ$）									
同时 JSF	仿射	—	2	83	162	490	245	1715	2450
（算法 3.48 和 3.50）	投影	—	2	83	162	1302	3	1317	1326
同时多点乘	仿射	2	10	78	163	482	241	1687	2410
（算法 3.48[i]）	投影	2	10	78	163	1222	11	1277	1310
交错 TNAF	仿射	6,4	18	60	163	448	224	1568	2240
（算法 3.51）	投影	6,4	18	60	163	1114	5	1139	1154
交错 TNAF	仿射	6,5	22	59	0[g]	164	59	459	636
（算法 3.51 和 3.69）	投影	6,5	22	59	0[g]	501	8	541	565

[a] 右列用域乘术语分别对于 $I/M=5$ 和 $I/M=8$ 给出计算开销。[b] 仿射。[c] 通过式(3.23)加。[d] 仅 x 坐标。[e] 开销 $A+M$。[f] 折半；设其开销为 $2M$。[g] 域运算包括对于 $S=M/7$ 应用τ。[h] $P+P\to P$。[i] 滑动窗口。

表 3.14　在 800 MHz Intel 奔腾 III 平台用通用寄存器进行点乘的时间，M 是奔腾分别在 I/M=80 和 I/M=8 的假设下二进制域和素域的乘法的估数。规格化给出了对于这一实现的等价 P-192 域乘法

NIST		域	PIII	800 MHz
曲	线　方　法	乘法 M	规格化 M	μs
未知点（kP，在线预计算）				
P-192	5-NAF（算法 3.36，w =5）	2016	2016	975
B-163	4-NAF（算法 3.36，w =4）	954	2953	1475
B-163	折半（算法 3.91，w =4）	687	2126	1050
K-163	5-TNAF（算法 3.70，w =5）	365	1130	625
固定基（kP，离线预计算）				
P-192	Comb 2-表（算法 3.45，w =4）	718	718	325
B-163	Comb（算法 3.44，w =5）	386	1195	575
K-163	6-TNAF（算法 3.70，w =6）	263	814	475
多点乘（$kP+lQ$）				
P-192	交错（算法 3.51，w =6,5）	2306	2306	1150
B-163	交错（算法 3.51，w =6,4）	1154	3572	1800
K-163	交错 TNAF（算法 3.51 和 3.69，w =6, 5）。	565	1749	1000

小结

列在表 3.12 和表 3.13 中的各种乘法的计算量不可以直接比较，因为即使是在相同的平台上分别实现二进制域和素域的乘法，其差别也是很大的。表 3.14 给出了在 800 MHz 的 Intel 奔腾 III 平台上的一种域乘法的实际运行时间和计算量。在此具体情况下，二进制域与素域的乘法时间的比率近似为 3.1（见 5.1.5 节），并且以 P-192 域的乘法为基准对乘法的运算量进行了规格化。

作为粗略的比较，运算时间表明对于未知点乘法，Koblitz 曲线（二进制）明显快于随机二进制曲线和素域曲线，这是因为在那里简单的域平方运算代替了大量的倍点运算。对于已知点的情况，预计算可以减少倍点的数量，并且快速的域乘法使 P-192 处于优势。对于计算 $kP+lQ$，其中仅对 kP 的预计算可以离线进行，K-163 和 P-192 的运行时间是可比较的，并且明显快于 B-163。

Koblitz 曲线的各种方法的运行时间比预期的要长，部分原因是因为寻找τ-adic NAF 的时间在评估中没有考虑（但是实际上它是包括在执行时间中的）。寻找τ-adic NAF 的算法 3.63 和算法 3.65 对于“大数”情况已用 OpenSSL 实现（见附录 C）。还要注意的是，可以通过采用交错技术有限地改善对于 Koblitz 曲线的已知点的计算（使用不多于 P-192 的预计算存储器）。

上述给出的比较还有一些局限。如在实现中仅仅使用了通用寄存器。工作站一般都有专有寄存器，可以用于提高域运算的速度。实际上，奔腾 III 处理器有浮点寄存器，可以加速素域的运算（见 5.1.2 节），并且单指令多数据（SIMD）寄存器特别有利于二进制域运算（见 5.1.3 节）。虽然所有的奔腾系列处理器都有 32×32 的整数乘法器，并都能给出 64 位的乘积，但是利用 P6 系列的处理器（如奔腾 III）的通用寄存器计算乘法，要快于早先的奔腾处理器，也快于最新的的奔腾 4 处理器。若实现平台的硬件乘法器比较弱或位数较短，则 P-192 的乘法就比 Koblitz 曲线的乘法难实现。在本书的多数章节，我们并没有区分数据关联存储与对于一个给定域或给定曲线的存储。本书没有讲述将大量存储器用于基于已知点的计算方法。

3.8 注释和进一步参考

3.1 节

Koblitz 的书[254]中的第 6 章，给出了椭圆曲线的简单介绍。对在椭圆曲线密码中用到的大量基本结论给出了证明的中级水平的教课书有 Charlap 和 Robbin 的[92, 93]，Enge 的[132]，Silverman 和 Tate 的[433]以及 Washington 的[474]。关于椭圆曲线理论的高级参考书有两本，都是由 Silverman 撰写的[429, 430]。

定理 3.8 取自 Waterhouse 的[475]，例 3.17 取自 Wittmann 的[484]。

3.2 节

D. Chudnovsky 和 C. Chudnovsky[96]研究了四种基本类型的椭圆曲线：(i) Weierstrass 型，$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$，本书始终采用这一类型；(ii) 雅可比型，$y^2 = x^4 + ax^2 + b$；(iii) 雅可比齐式，把椭圆曲线表示为两个二次项 $x^2 + y^2 = 1$ 和 $k^2x^2 + z^2 = 1$之交；(iv) Hessian 齐式：$x^3 + y^3 + z^3 = Dxyz$。Liardet 和 Smart[291]观察到在雅可比齐式下点加和倍点的运算规则相同，所以可以增强抵抗能量分析攻击的能力。Joye 和 Quisquater[231]指出这种性质对于 Hessian 齐式也成立，并且给出结论：Hessian 齐式的加法所需要的运算比雅可比齐式的加法少（12 次乘对 16 次乘）。Smart[442]观察到在 Hessian 齐式下椭圆曲线群运算的对称性使得点加和倍点的计算可以并行处理（用三个处理器）。注意，定义在 $\mathbb{F}_q$ 上的 Hessian 齐式椭圆曲线有理数点群一定包含一个 3 阶的点，因此 Hessian 齐式不能用于 NIST 的椭圆曲线标准。Frium[151]对 Hessian 齐式的椭圆曲线做了广泛的研究。

D. Chudnovsky 和 C. Chudnovsky[96]提出了 Chudnovsky 坐标。Cohen, Miyaji 和 Ono[100]对混合坐标系的不同组合进行了比较。注意，他们改进的雅可比坐标对于椭圆曲线 $y^2=x^3+ax+b$（其中 $a=-3$）的点加和倍点不能产生加速作用，然而这种思想对于在算法 3.23 中的重复倍点运算有加速作用。Lim 和 Hwang[293]选择了相应于 $(X/Z^2, Y/2Z^3)$ 的投影坐标，使得用 2 除被省略了，而点加却需要两个附加的域加法。

López 和 Dahab[300]提出了 LD 坐标。运算公式反映了 Lim 和 Hwang[294]，Al-Daoud，Mahmod，Rushdan，Kilicman[10]等人对此所做的改进，这种改进使得混合坐标的点加只需要一次乘法（和一次附加的平方）。

若域的乘法是采用与算法 2.36 类似的方法（带有数据相关的预计算），则 King[246]建议认真组织点的算术运算以减少预计算的数量（即一个预计算表可以多次使用）。取决于存储器的限制，可使用预计算的一个保存的表，也可使用多个保存的表。

计算重复倍点的算法 3.23 是一个可改进的例子。King[246]给出了一种改进，用于实现由 López 和 Dahab[300]提出的点加和倍点计算。在计算 $2(P+Q)$ 时，一次域乘法可以转换为两次平方运算，因为相加时所需要的值 X_3Z_3 可以用于后面的倍点计算。由 Eisenträger，Lauter 和 Montgomery[129]给出的建议与此类似，它在用仿射坐标计算 $2P+Q$ 时省去了域乘法（在计算中间值 $P+Q$ 时省掉 y 坐标的计算，注意 $2P+Q=(P+Q)+P$。

3.3 节

Knuth[249]描述了从右向左的二进制方法，并扩展成 m 进制的方法。Cohen[99]讨论了基为 2^k 的从右向左和从左向右的方法。Gordon[179]提供了一种有用的求幂方法。Menezes，van Oorschot 和 Vanstone[319]建议了一种比现有求幂方法更具有一般性的求幂算法。

定理 3.29 中的密度概念是由 Morain 和 Olivos[333]给出的。窗口 NAF 方法（算法 3.35 和算法 3.36）是由 Solinas[446]提出的，他注意到“更精心设计的窗口方法是存在的（见[179]），但这需要大量的初始计算并且很难比现有方做得更好”。

Möller[329]提出了一种碎片窗口技术，扩展了滑动窗口方法和窗口 NAF 方法。这种方法在预计算量方面具有更大的弹性，当存储空间受限时这有实际意义（参见注 3.39）。对于窗口宽度 $w>2$ 和给定的奇参数 $v\leqslant 2^{w-1}-3$，碎片窗口表示法的平均密度为 $1/(w+1+\frac{v+1}{2^{w-1}})$；碎片窗口方法在有效窗口大小方面比宽度 w-NAF 方法相比增加了 $\frac{v+1}{2^{w-1}}$。

算法 3.40 是由 López 和 Dahab[299]提出的。这一算法的基础是 Montgomery 方法[331]。Okeya 和 Sakurai[359]把这一工作推扩到了特征大于 3 的有限域上的椭圆曲线。

固定基窗口方法（算法 3.41）是由 Brickell, Gordon, McCurley 和 Wilson[72]提出的。

Gordon[179]引证了 de Rooij 的论文[109]以及 Lim 和 Lee 的论文[295]的向量加法链方法，这些论文表明了如下的意见："BGMW 方法使用的存储器太多"。算法 3.44 和算法 3.45 是 Lim-Lee 方法的一种特例，更一般的情况在注 3.47 中给出了介绍。

在注 3.46 中 Lim-Lee 方法采用了同时加技术，这一技术是由 Lim 和 Hwang 在[294]中提出的。对组合参数 $v > 2$ 的增强措施（见注 3.47）把求逆的数量从 $v-1$ 减少到 $\lceil \text{lb } v \rceil$（$\lfloor v/2 \rfloor e$ 个元素需要临时存储）。

对同时点乘的"Shamir 欺骗"（算法 3.48）是由 ElGamal[131]提供给 Shamir 的。应用滑动窗口方法进行改进是由 Yen, Laih 和 Lenstra 提出的[487]。联合稀疏形是由 Solinas[447]提出的。Proos[383]把联合稀疏形方法扩展到了对任意数量的整数。"零列搜索法"也被提出来，并用带符号的二进制表示法扩展了 Lim-Lee 方法，以增加指数矩阵中零列的数量。这一改进（类似量的存储）取决于点加与倍点的相对开销以及预计算的存储量；若点加与倍点有相同的开销，则对于 160 位的 k 和不多于 32 个点的预计算的实例，在计算 kP 的过程中可减少近似 10%的点运算。

交错技术（算法 3.51）是由 Gallant, Lambert, Vanstone[160]和 Möller[326]提出的。Möller[326]注意到计算 kP 的交错方法，那里 k 被分裂开，然后对小段采用 w-NAF 方法，但可能会浪费部分 w-NAF。于是产生了窗口 NAF 分裂方法，这对于 w 较大时特别有意义。其基本思想是首先计算 k 的 w-NAF，然后再进行分裂。

3.4 节

一些椭圆曲线之所以称为 Koblitz 曲线，是因为 Koblitz 在文献[253]中首先建议将其用于密码。Koblitz 讲明了对于 Koblitz 曲线上的点计算 kP，整数 k 的 τ-adic 表示法如何能够省去倍点运算。Meier 和 Staffelbach[312]表明了如何通过首先在 $\mathbb{Z}[\tau]$ 中对 k 模 $\tau^m - 1$ 执行约减运算来获得 k 的短 τ-adic 表示。TNAF 方法和 w TNAF 方法都是由 Solinas 提出的[446]。Park, Oh, Lee, Lim 和 Sung[370]提出了另一种获得短的 τ-adic 表示法的方法。这一方法把 τ-adic 表示的长度减少了 lb h，因此当余因子 h 很大时这种方法的作用就很大。

Müller [334]将 Koblitz 曲线的快速点乘技术推广到了定义在某些小的二进制域（如 $\mathbb{F}_{2^2}$, $\mathbb{F}_{2^3}$, $\mathbb{F}_{2^4}$ 和 $\mathbb{F}_{2^5}$）上的椭圆曲线，Koblitz[256]和 Smart[439]又将其推广到了定义在某些具有奇数特征的小扩域上的椭圆曲线。Günther, Lange 和 Stein[185]提出亏格为 2 的超椭圆曲线雅可比坐标的点乘一般方法，其研究主要集中于定义在 $\mathbb{F}_2$ 上的曲线 $y^2 + xy = x^5 + 1$ 和 $y^2 + xy = x^5 + x^2 + 1$。他们的方法又被 Choie 和 Lee[94]推广到了定义在任意特征的有限域上的亏格为 2，3 和 4 的超椭圆曲线上。

3.5 节

Gallant, Lambert 和 Vanstone[160]开拓了加速椭圆曲线的点乘的有效可计算自同态的方法，他们还提出了算法 3.74 用于计算平衡长度 2 的乘数表示法。在例 3.73 中的 P-160 曲线取自无线 TLS 规范[360]。例 3.76 是由 Solinas 给出的[447]。

Sica, Ciet 和 Quisquater[428]证明了在算法 3.74 中向量 $v_2 = (a_2, b_2)$具有小的范数。Park, Jeong, Kim 和 Lim[368]提出了计算平衡长度 2 的乘数表示法的一种替代方法，并且证明了他们的方法总是起作用的。他们的经验表明，他们提出的这种分解方法的性能与算法 3.74 的性能实际上是相同的。另一种方法是由 Kim 和 Lim 提出的[242]。Müller[335]将 Gallant-Lambert-Vanstone 方法推广到了平衡长度 m 乘数表示法，并且证明对于定义在最佳扩域上的特定椭圆曲线的点乘有加速效果。Park, Jeong 和 Lim[369]又将其推广到了具有有效可计算自同态的超椭圆曲线。

Ciet, Lange, Sica 和 Quisquater[98]把 Koblitz 曲线的 τ-adic 展开技术扩展到了具有范数超过 1 的自同态 ϕ 的素域上的椭圆曲线上。与 Gallant-Lambert-Vanstone 方法相比，在计算 kP 中近似(lb n)/2 的倍点计算替换为双倍应用 ϕ 的运算。对于特征多项式为 $x^2 \pm x + 2$的自同态 ϕ，一种推广把联合稀疏形（见 3.3.3 节）扩充到了 ϕ-JSF。

3.6 节

点的折半技术分别由 Knudsen[247]和 Schroeppel[413]独立给出。Fong, Hankerson, López 和 Menezes[144]给出了折半技术与基于倍点的方法的比较。

在计算点乘 kP 且 P 预先知道的情况下，点折半方法的优点是明显的，较小的乘法与求逆之比使折半方法受益。Knudsen 的分析[247]指出，在 $I/M \approx 3$时，对于未知点情况，折半方法有 39%的收益。Fong, Hankerson, López 和 Menezes[144]指出，对于 SPARC 和奔腾平台这一比率是非常优的，对于这些平台，最快的时间给出 $I/M > 8$。类似的分析指出，大的 I/M 比率将导致在未知点情况下收益近似下降 25%。若 P 是事先知道的且具有适当的预计算存储空间，则基于折半的方法就不是最好的。对于计算 $kP + lQ$，其中仅 P 是预先知道的，则基于折半的方法与基于倍点的方法相比差别是很小的，对于通常的 I/M 比率，折半方法稍快一些。

算法 3.91 致力于解决 Knudsen[247]提出的挑战，以导出“一种有效的投影坐标折半算法”。然而这一算法并不能对投影点进行折半，但它说明了对于折半和投影坐标的一种有效的窗口方法，特别适用于大 I/M 的情况。计算 Q 的脚注 3 取自 Knuth[249，习题 4.6.3-9]，还可参考 Möller 的文献[326, 329]。

3.7 节

用于表 3.14 的实现细节将在 5.1.5 节中介绍。简单地说，仅仅使用了通用寄存器，素域的算术运算主要用汇编实现，而二进制域的算术运算主要用 C 实现，除了一段关于多项式次数的计算外。编程语言为 Intel 编译器版本 6 以及 Netwide 汇编器（NASM），系统平台为 Intel 奔腾 III 平台，运行的是 Linux2.2 操作系统。

粗略地认为 32 位的 Intel 奔腾 III 属于工作站类处理器，而其他流行的处理器还有 DEC Alpha 64 位处理器族和 Sun SPARC 32 位和 64 位处理器族等。Lim 和 Hwang[293, 294]对于 Intel 奔腾 II 和 DEC Alpha 处理器，给出了扩域（特别是最佳扩域 OEF）和曲线的计算时间分析。Smart[440]给出了 Sun UltraSPARC IIi 和 Intel 奔腾 Pro 对于素域、二进制域、最佳扩域上的椭圆曲线的计算时间比较。Hankerson, López 和 Menezes[189]以及 Brown, Hankerson, López 和 Menezes[77]集中研究了 Koblitz 曲线在 Intel 奔腾 II 环境上的域和曲线计算时间分析。De Win, Mister, Preneel 和 Wiener[111]比较了在 Intel 奔腾 Pro 环境上的 ECDSA，DSA 和 RSA 签名算法的计算时间分析。Weimerskirch, Stebila 和 Chang Shantz[478]讨论了对于任意的域大小和任意的约减多项式的二进制域的实现，并对于 Intel 奔腾 III 和 32 位、64 位代码的 Sun UltraSPARC III 环境给出了计算时间分析。

工作站上的专用硬件可以极大地提高计算速度。Bernstein[43]对于 $\mathbb{F}_p(p=2^{224}-2^{96}+1)$ 上的 NIST 曲线给出了点乘的计算时间分析，计算环境分别为 AMD，DEC，Intel 和 Sun 处理器的浮点硬件设备，见 http:w//cr.yp.to/nistp224/timings.html。5.1 节给出了浮点硬件和 SIMD 硬件的概况介绍。

第4章 密码协议

本节介绍基于椭圆曲线的签名方案、公钥加密算法和密钥建立方案。4.1 节纵览目前已知求解椭圆曲线离散对数问题的算法，这对于所有基于椭圆曲线的密码方案的安全性都是至关重要的。4.1 节还将简要介绍基于椭圆曲线的 Diffie-Hellman 问题，并讨论在一些协议的安全性证明中作为前提假设的判定性 Diffie-Hellman 问题。4.2 节和 4.3 节将讨论在椭圆曲线协议中使用的参数组以及密钥对的生成和确认。4.4 节、4.5 节和 4.6 节分别介绍 ECDSA 和 EC-KCDSA 签名方案，ECIES 和 PSEC 公钥加密方案，以及 STS 和 ECMQV 密钥建立方案。4.7 节为注释和参考文献。

4.1 椭圆曲线离散对数问题

椭圆曲线离散对数问题的困难性是所有椭圆曲线密码方案安全性的基础。

定义 4.1 椭圆曲线离散对数问题（ECDLP）：给定定义于有限域 $\mathbb{F}_q$ 上的椭圆曲线 E，基点 $P \in E(\mathbb{F}_q)$，阶为 n，点 $Q \in \langle P \rangle$，寻找一个整数 $l \in [0, n-1]$ 使得 $Q = lP$。整数 l 称为 Q 的基于 P 的离散对数，表示为 $l = \log_P Q$。

为了抵抗所有已知的对 ECDLP 的攻击，密码方案中使用的椭圆曲线的参数应该谨慎选择。最简单的求解 ECDLP 问题的算法是穷举搜索：连续计算一系列点 $P, 2P, 3P, 4P, \ldots$，直到得到 Q。该方法的运行时间最坏情况下为 n 步，平均情况下为 $n/2$ 步。因此可以通过选择参数 n 足够大的椭圆曲线，使得穷举攻击的计算量不可实际实现（如 $n \geqslant 2^{80}$）。对于 ECDLP，已知最好的通用攻击方法是将 Pohlig-Hellman 算法和 Pollard's rho 算法相结合，该方法的运行时间为完全指数时间 $O(\sqrt{p})$，其中 p 是 n 的最大素因子。为了抵抗该种攻击，椭圆曲线参数的选择要求 n 含有大素数因子 p，p 应该足够大，以使得 $\sqrt{p}$ 的计算量不可实际实现（如 $p > 2^{160}$）。另外，椭圆曲线的参数选择还要防止所有其他的已知攻击方法（见 4.1.4 节），这样所选得的 ECDLP 才会被认为对于当今的计算水平是难解的。

需要注意的是，ECDLP 的难解并没有数学上的证明。目前还没有证明不存在求解

ECDLP 的有效算法。事实上，得出这样的证明将会是轰动的结果。例如，若能证明 ECDLP 不存在多项式时间算法，则就意味着证明了 $P \neq NP$，而后者是计算机科学的基本理论中著名的公开问题[①]。而且没有任何的理论证明 ECDLP 是难解的，例如不知道 ECDLP 是否是 NP 困难的[②]，并且不太可能证明其为 NP 困难的，因为已知 ECDLP 同时属于 NP 和 co-NP[③]。

然而，通过许多年的考验说明了 ECDLP 的难解性。首先，自从 1985 年椭圆曲线密码被提出以来，该问题被许多学者研究，到目前还没有发现亚指数时间的通用算法。其次，Shoup 已经证明了求解阶为素数 n 的一般群的离散对数问题下界为 $\sqrt{n}$，这里群的元素为随机的比特串，并且只能通过假定的预言器进行群运算。虽然 Shoup 的结论不能得出求解 ECDLP 确实是困难的（因为椭圆曲线群的元素有特殊含义并且不是随机表示的），但是这给出了从理论上证明对一些群的离散对数问题是难解的希望。

在 4.1.1 节和 4.1.2 节中将分别给出求解 ECDLP 的 Pohlig-Hellman 算法和 Pollard's rho 算法。在 4.1.3 节中，我们将研究攻击 ECDLP 的亚指数时间非通用算法。同构攻击将 ECDLP 约减为一个同构群上的 DLP，而 DLP 已知有亚指数时间（或更快的）算法。这些攻击包括 Weil 和 Tate 配对攻击，针对素域异常曲线的攻击，以及 Weil 下降方法。虽然这类同构攻击的数学理论相对较深，但是它们的密码学含义都很容易理解，并且已知有简单的对策使得给定的椭圆曲线可证明免于此类攻击。由此，在 4.1.4 节中我们将讨论如何防止此类同构攻击成功，并给出该攻击的密码学意义及相应对策，并给出此类攻击的详细数学描述。最后，4.1.5 节讨论使用 ECDLP 的两个密码学问题，分别是椭圆曲线 Diffie-Hellman 问题（ECDHP）和椭圆曲线判定性 Diffie-Hellman 问题（ECDDHP）。

4.1.1 Pohlig-Hellman 攻击

Pohlig-Hellman 算法有效地将计算 $l = \log_P Q$ 约减为计算 $\langle P \rangle$ 的素数阶子群的离散对数问题。这说明求解 $\langle P \rangle$ 上的 ECDLP 不比求解素数阶子群的 ECDLP 困难。因此，为了最大可能地抵抗 Pohlig-Hellman 攻击，椭圆曲线参数的选择应使得 P 的阶 n 含有大素数因子。下面我们给出 Pohlig-Hellman 算法。

① 判定问题（YES/NO）中存在多项式时间算法的一类问题称为 P 类问题。而 NP 类问题是指可满足问题中如果给出合适的证明，其YES 解可以在多项式时间内验证的判定问题。显然 $P \subseteq NP$，但至今未知是否有 $P = NP$。

② 对一个问题，若 NP 中的所有问题都可在多项式时间内约减到它，则称其为 NP 困难的。若对一个 NP 完全问题找到多项式时间算法，则可以证明 $P = NP$。

③ co-NP 类问题是指若给出合适的证明，其NO解可以在多项式时间内验证的判定问题。至今未知是否有 $NP = co\text{-}NP$。若能找出一个 NP 困难的问题同时属于 NP 和 co-NP，则可以推出 $NP = co\text{-}NP$。

设 n 的因子分解为 $n=p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$。Pohlig-Hellman 方法是对每个$1\leqslant i\leqslant r$，计算 $l_i=l \bmod p_i^{e_i}$，再对$l\in[0,n-1]$求解同余方程组（中国剩余定理保证其存在惟一解）：

$$\begin{aligned} l &\equiv l_1 \pmod{p_1^{e_1}} \\ l &\equiv l_2 \pmod{p_2^{e_2}} \\ &\vdots \\ l &\equiv l_r \pmod{p_r^{e_r}} \end{aligned}$$

现在我们看通过计算每个 l_i 如何将其约减为计算$\langle P\rangle$的阶为 p_i 的子群的离散对数 e_i。为了便于表示，我们用 p 表示 p_i，用 e 表示 e_i。令 l_i 以 p 为基数的表示为

$$l_i=z_0+z_1p+z_2p^2+\cdots+z_{e-1}p^{e-1},$$

其中每个 $z_i\in[0,p-1]$。每次计算 $z_0,z_1,\ldots,z_{e-1}$ 中的一个。首先计算$P_0=(n/p)P$ 和 $Q_0=(n/p)Q$。因为 P_0的阶为 p，可得

$$Q_0=\frac{n}{p}Q=l\left(\frac{n}{p}P\right)=lP_0=z_0P_0。$$

所以 $z_0=\log_{P_0}Q_0$ 可以通过求解$\langle P_0\rangle$的 ECDLP 得到。下一步，我们计算 $Q_1=(n/p^2)(Q-z_0P)$。我们有

$$\begin{aligned} Q_1&=\frac{n}{p^2}(Q-z_0P)=\frac{n}{p^2}(l-z_0)P=(l-z_0)\left(\frac{n}{p^2}P\right) \\ &=(z_0+z_1p-z_0)\left(\frac{n}{p^2}P\right)=z_1\left(\frac{n}{p}P\right)=z_1P_0。 \end{aligned}$$

所以 $z_1=\log_{P_0}Q_1$ 可以通过求解 $\langle P_0\rangle$ 的 ECDLP 得到。通常，若 $z_0,z_1,\ldots,z_{t-1}$都被计算出来，则 $z_t=\log_{P_0}Q_t$，其中

$$Q_t=\frac{n}{p^{t+1}}\left(Q-z_0P-z_1pP-z_2p^2P-\cdots-z_{t-1}p^{t-1}P\right)。$$

例 4.2 （用 Pohlig-Hellman 算法求解 ECDLP）考虑定义于有限域$\mathbb{F}_{7919}$上的椭圆曲线 E：

$$E:y^2=x^3+1001x+75。$$

令$P=(4023,6036)\in E(\mathbb{F}_{7919})$。$P$的阶为

$$n=7889=7^3\cdot 23。$$

令 $Q=(4135,3169)\in\langle P\rangle$。求解 $l=\log_P Q$。

(i) 首先计算 $l_1=l \bmod 7^3$。将 l_i 写做 $l_1=z_0+z_1 7+z_2 7^2$，计算

$$P_0=7^2 23P=(7801,2071)$$
$$Q_0=7^2 23Q=(7801,2071)$$

并算得 $Q_0=P_0$；因此 $z_0=1$。接下来计算

$$Q_1=7\cdot 23(Q-P)=(7285,14)$$

并算得 $Q_1=3P_0$；因此 $z_1=3$。最后计算

$$Q_2=23(Q-P-3\cdot 7P)=(7285,7905)$$

并算得 $Q_2=4P_0$；因此 $z_2=4$。于是可得 $l_1=1+3\cdot 7+4\cdot 7^2=218$。

(ii) 接下来计算 $l_2=l \bmod 23$。计算

$$P_0=7^3 P=(7190,7003)$$
$$Q_0=7^3 Q=(2599,759)$$

并算得 $Q_0=10P_0$；因此 $l_2=10$。

(iii) 最后求解同余方程组

$$l\equiv 218 \pmod{7^3}$$
$$l\equiv 10 \pmod{23}$$

并算得 $l=4334$。

对于 4.1 节剩余的部分，我们假定 P 的阶 n 为素数。

4.1.2 Pollard's rho 攻击

Pollard's rho 算法的主要思想是寻找模 n 的不同 (c',d') 和 (c'',d'')，使得

$$c'P+d'Q=c''P+d''Q。$$

于是

$$(c'-c'')P=(d''-d')Q=(d''-d')lP$$

且有

$$(c'-c'')\equiv(d''-d')l \pmod{n}。$$

于是$l=\log_P Q$可以通过计算

$$l=(c'-c'')(d''-d')^{-1} \bmod n \tag{4.1}$$

来得到。

一种寻找这样的(c',d')和(c'',d'')的直接方法是随机选择整数$c,d\in[0,n-1]$，建立三元表$(c,d,cP+dQ)$并以第三项排序，直到第二次找到同样的$cP+dQ$——这称为一个碰撞。通过生日悖论[①]，找到一个碰撞预计需要的迭代次数是$\sqrt{\pi n/2}\approx 1.2533\sqrt{n}$。该算法的缺点是需要$\sqrt{\pi n/2}$个三元表的存储空间。

Pollard's rho算法寻找(c',d')和(c'',d'')的预计时间大概和直接的方法相当，但其需要的存储空间近于零。其思想是定义一个迭代函数$f:\langle P\rangle\to\langle P\rangle$，使得给定$X\in\langle P\rangle$和$c,d\in[0,n-1]$且$X=cP+dQ$，容易计算$\overline{X}=f(X)$和$\overline{c},\overline{d}\in[0,n-1]$且$\overline{X}=\overline{c}P+\overline{d}Q$。此外，函数$f$还要具有随机性。

下面是一个合适的迭代函数的例子。“随机地”将$\langle P\rangle$分为大致相等的L个集合$\{S_1,S_2,\ldots,S_L\}$。L通常的取值为16和32。例如，若$L=32$，对于点$X\in\langle P\rangle$，若X的x坐标的右边5比特的整数表示为$j-1$，就将X分入集合S_j。若$X\in S_j$，则记为$H(X)=j$，并称H为分割函数。最后，令$a_j,b_j\in_R[0,n-1]$，其中$1\leqslant j\leqslant L$。则$f:\langle P\rangle\to\langle P\rangle$为

$$f(X)=X+a_jP+b_jQ,\ \text{其中}\ j=H(X)。$$

显然，若$X=cP+dQ$，则$f(X)=\overline{X}=\overline{c}P+\overline{d}Q$，其中$\overline{c}=c+a_j \bmod n$，$\overline{d}=d+b_j \bmod n$。

现在任意点$X_0\in\langle P\rangle$确定一个点序列$\{X_i\}_{i\geqslant 0}$，其中对于$i\geqslant 1$有$X_i=f(X_{i-1})$。因为集合$\langle P\rangle$是有穷的，序列终将产生碰撞并陷入循环；也就是说，对于$s\geqslant 1$，存在最小的索引t使得任意的$X_t=X_{t+s}$，且对所有的$i\geqslant t+s$有$X_i=X_{i-s}$（见图4.1）。这里t称为序列的尾长度，s称为序列的循环长度。若假设f为随机函数，则序列出现首次碰撞的预计步数为$\sqrt{\pi n/2}$。此外，预计的尾长度为$t\approx\sqrt{\pi n/8}$，循环长度为$s\approx\sqrt{\pi n/8}$。

一个碰撞，即对于点X_i,X_j有$X_i=X_j$且$i\neq j$，可以用Floyd循环查找算法来计算。该算法对于$i=1,2,3\ldots$计算点对(X_i,X_{2i})直到$X_i=X_{2i}$。计算出新的点对以后可以将老的点对丢弃，因此其需要的存储空间近于零。预计得到$X_i=X_{2i}$之前计算这些点对的次数k显然满足$t\leqslant k\leqslant t+s$。事实上，假定$f$是随机函数，则预计值$k$约为$1.0308\sqrt{n}$，因此预计的

① 假设罐子里有编号从1到n的n个小球。每次随机从罐中取一个小球并放回。则预计出现同一个小球第二次被取出的总抽取次数约为$\sqrt{\pi n/2}$。若$n=365$，每个小球代表一年中的每一天，则上述内容可理解为要想同一房间中的人中至少有两个人的生日相同，那么房间的总人数约为$\sqrt{\pi 365/2}\approx 24$。这个数字小得令人吃惊，于是被称为“生日悖论”。

椭圆曲线群运算的次数约为$3\sqrt{n}$。整个算法步骤见算法 4.3。注意算法失败（例如在步骤 7 中 $d' = d''$）的概率趋近于零。

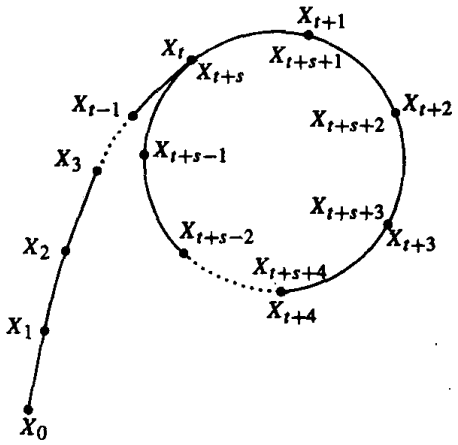


图 4.1 Pollard's rho 算法的 ρ型序列$\{X_i\}$，其中 t 为尾长度，s 为循环长度

算法 4.3 求解 ECDLP 的 Pollard's rho 算法（单处理器）

输入：$P \in E(\mathbb{F}_q)$，P 的阶 n 为素数，$Q \in \langle P \rangle$。

输出：离散对数$l = \log_P Q$。

1. 选择集合数目 L（如$L = 16$或$L = 32$）。
2. 选择一个分割函数$H : \langle P \rangle \to \{1, 2, \ldots, L\}$。
3. 对于 j 从 1 到 L，执行

 3.1 选择$a_j, b_j \in_R [0, n-1]$。

 3.2 计算 $R_j = a_j P + b_j Q$。
4. 选择$c', d' \in_R [0, n-1]$并计算$X' = c'P + d'Q$。
5. 令$X'' \leftarrow X', c'' \leftarrow c', d'' \leftarrow d'$。
6. 重复如下操作：

 6.1 计算$j = H(X')$。

 令$X' \leftarrow X' + R_j, c' \leftarrow c' + a_j \bmod n, d' \leftarrow d' + b_j \bmod n$。

 6.2 对于 i 从 1 到 2，执行

 计算$j = H(X'')$。

 令$X'' \leftarrow X'' + R_j, c'' \leftarrow c'' + a_j \bmod n, d'' \leftarrow d'' + b_j \bmod n$。

 直到$X' = X''$。
7. 若$d' = d''$，则返回（“算法失败”）；

 否则，计算$l = (c' - c'')(d'' - d')^{-1} \bmod n$并返回$(l)$。

例 4.4 （用 Pollard's rho 算法求解 ECDLP）考虑定义于有限域$\mathbb{F}_{229}$上的椭圆曲线 E：

$$E: y^2 = x^3 + x + 44。$$

点 $P = (5, 116) \in E(\mathbb{F}_{229})$。$P$ 的阶为素数 $n = 239$。令 $Q = (155, 166) \in \langle P \rangle$。求解 $l = \log_P Q$。

我们选择分割函数 $H: \langle P \rangle \to \{1, 2, 3, 4\}$的集合数目为 $L = 4$：

$$H(x, y) = (x \bmod 4) + 1,$$

四个三元表为

$$[a_1, b_1, R_1] = [79, 163, (135, 117)]$$
$$[a_2, b_2, R_2] = [206, 19, (96, 97)]$$
$$[a_3, b_3, R_3] = [87, 109, (84, 62)]$$
$$[a_4, b_4, R_4] = [219, 68, (72, 134)]。$$

下表列出了算法 4.3 的步骤 4 中$(c', d') = (54, 175)$时计算的三元表(c', d', X')和(c'', d'', X'')。

迭代次数	c'	d'	X'	c''	d''	X''
–	54	175	(39,159)	54	175	(39,159)
1	34	4	(160, 9)	113	167	(130,182)
2	113	167	(130,182)	180	105	(36, 97)
3	200	37	(27, 17)	0	97	(108, 89)
4	180	105	(36, 97)	46	40	(223,153)
5	20	29	(119,180)	232	127	(167, 57)
6	0	97	(108, 89)	192	24	(57,105)
7	79	21	(81,168)	139	111	(185,227)
8	46	40	(223,153)	193	0	(197, 92)
9	26	108	(9, 18)	140	87	(194,145)
10	232	127	(167, 57)	67	120	(223,153)
11	212	195	(75,136)	14	207	(167, 57)
12	192	24	(57,105)	213	104	(57,105)

算法发现$192P + 24Q = 213P + 104Q$，因此$l = (192 - 213) \cdot (104 - 24)^{-1} \bmod 239 = 176$。

并行 Pollard's rho 攻击

假定现在有 M 台处理器可用于求解 ECDLP。一种直接的方法是在每台处理器上（随机选择不同的起始点 X_0）独立运行 Pollard's rho 算法，直到其中任意一台处理器得出结果。分析表明在得出结果之前，每台处理器执行椭圆曲线运算的预计次数约为 $3\sqrt{n/M}$。加速比因子仅为$\sqrt{M}$。

Van Oorschot 和 Wiener 提出了 Pollard's rho 算法的一个变体，使得当使用 M 台处理器

时的加速比因子为 M。其基本思想是使处理器产生的序列$\{X_i\}_{i\geqslant 0}$同其他序列产生碰撞。详细方法是，每台处理器随机选择自己的起始点X_0，但所有的处理器使用同样的迭代函数 f 来计算接下来的点X_i。于是，若由两台不同处理器生成的序列在某点碰撞，如图 4.2 所示，则这两个序列从该点以后相同。

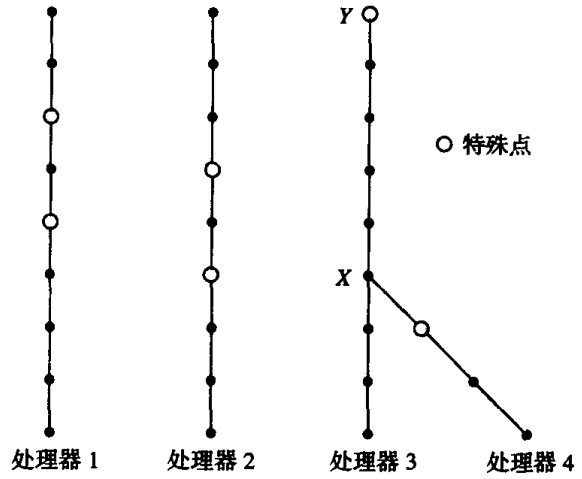


图 4.2 并行 Pollard's rho 算法生成的序列。由处理器 3 和 4 产生的序列首先在 X 碰撞。算法得出在 Y 碰撞，也就是 X 之后的第一个特殊点

Floyd 循环查找算法可用来找出由单处理器生成的序列的碰撞。下面的方法可以有效地从不同处理器生成的序列中找出碰撞。首先选择特殊的易于检测的点，例如 x 坐标的前 t 位都为零。令 θ 为$\langle P\rangle$中这样的特殊点所占的比例。当任一处理器遇到特殊点后，将其传送到中心服务器经过排序的表中。当服务器第二次收到同样的特殊点后，即可通过式(4.1)计算出所求的离散对数，并通知所有处理器停机。找到碰撞预计平均每台处理器所执行的次数为$(\sqrt{\pi n/2})/M$。全部特殊点造表约需要$1/\theta$步。因此找到碰撞预计每台处理器所执行的椭圆曲线运算的次数约为

$$\frac{1}{M}\sqrt{\frac{\pi n}{2t}}+\frac{1}{\theta}, \tag{4.2}$$

且该并行化的 Pollard's rho 算法的加速比同处理器的数目呈线性关系。显然处理器之间不需要通信，仅需要同服务器进行少量通信。此外，中心服务器上需要的总存储空间可通过选择特殊点的性质来控制。整个算法步骤见算法 4.5。注意算法失败(例如在步骤 7 中 $d'=d''$)的概率趋近于零。

算法 4.5 求解 ECDLP 的并行 Pollard's rho 算法

输入：$P \in E(\mathbb{F}_q)$，P 的阶 n 为素数，$Q \in \langle P \rangle$。

输出：离散对数 $l = \log_P Q$。

1. 选择集合数目 L（如 $L = 16$ 或 $L = 32$）。
2. 选择一个分割函数 $H : \langle P \rangle \to \{1, 2, \ldots, L\}$。
3. 在 $\langle P \rangle$ 中选择特殊点的性质。
4. 对于 j 从 1 到 L，执行
 4.1 选择 $a_j, b_j \in_R [0, n-1]$。
 4.2 计算 $R_j = a_j P + b_j Q$。
5. 每台处理器 M 进行如下操作：
 5.1 选择 $c, d \in_R [0, n-1]$ 并计算 $X = cP + dQ$。
 5.2 重复如下操作：
 若 X 为特殊点，则发送 (c, d, X) 到中心服务器。
 计算 $j = H(X)$。
 令 $X \leftarrow X + R_j$，$c \leftarrow c + a_j \bmod n$，且 $d \leftarrow d + b_j \bmod n$。
 直到服务器第二次收到同样的特殊点 Y。
6. 令与 Y 相关的两个三元组为 (c', d', Y) 和 (c'', d'', Y)。
7. 若 $d' = d''$，则返回（"算法失败"）；
 否则，计算 $l = (c' - c'')(d'' - d')^{-1} \bmod n$ 并返回 (l)。

用自同构加速 Pollard's rho 算法

令 $\psi : \langle P \rangle \to \langle P \rangle$ 为 P 的自同构，其中 $P \in E(\mathbb{F}_q)$ 的阶为 n。假定 ψ 的计算量大大少于椭圆点加的计算量。设 ψ 的阶为 t，即 t 是对所有 $R \in \langle P \rangle$ 使得 $\psi^t(R) = R$ 的最小正整数。$\langle P \rangle$ 上的等价关系~定义为

$$R_1 \sim R_2，\text{当且仅当存在 } j \in [0, t-1]，\ R_1 = \psi^j(R_2)。$$

包含点 $R \in \langle P \rangle$ 的等价集 $[R]$ 为

$$[R] = \{R, \psi(R), \psi^2(R), \ldots, \psi^{l-1}(R)\},$$

其中 l 是 t 的最小除数，满足 $\psi^l(R) = R$。

为了加速算法，可以改进迭代函数 f，使得其定义在等价集上（而不再只是定义在 $\langle P \rangle$ 中的点上）。为了达到这个要求，我们为每个等价集 $[R]$ 定义规范的表示 $\overline{R}$。例如，$\overline{R}$ 可以定

义为在[R]中 x 坐标最小（把 y 坐标看做整数）的点。于是我们在规范表示上定义迭代函数 g 为

$$g(R)=\overline{f(R)}。$$

假设我们现在知道整数$\lambda\in[0,n-1]$满足

$$\psi(P)=\lambda P,$$

那么因为ψ是群同构，我们对于所有$R\in\langle P\rangle$有$\psi(R)=\lambda R$。于是若我们知道整数 a 和 b 使得$X=aP+bQ$，则我们就能快速计算出 a' 和 b' 使得$\overline{X}=a'P+b'Q$。若$\overline{X}=\psi^j(X)$，则有 $a'=\lambda^j a \bmod n$和$b'=\lambda^j b \bmod n$。

现在可以把 g 作为并行 Pollard's rho 算法中的迭代函数。序列的起始点为$X'_0=\overline{X_0}$，其中$X_0=a_0P+b_0Q$且$a_0,b_0\in_R[0,n-1]$。序列中接下来的点可以通过迭代计算：对于$i\geqslant 1$有$X'_i=g(X'_{i-1})$。设大部分的等价集的大小为 t，则搜索空间约为n/t（若不使用等价集，则为 n），于是改进后的并行 Pollard's rho 算法的期望运行时间为

$$\frac{1}{M}\sqrt{\frac{\pi n}{2t}}+\frac{1}{\theta}, \tag{4.3}$$

与式(4.2)相比，速度提高了$\sqrt{t}$。

例 4.6 （使用负映射）负映射$\psi(P)=-P$的阶为 2，其需要的特征如上所述。于是使用负映射下的等价集时，并行 Pollard's rho 算法的期望运行时间为

$$\sqrt{\frac{\pi n}{2M}}+\frac{1}{\theta}。 \tag{4.4}$$

与式(4.2)相比，该方法的速度提高了$\sqrt{2}$并且可以应用到所有椭圆曲线。

例 4.7 （对 Koblitz 曲线改进 Pollard's rho 算法）3.4 节中讨论的 Koblitz 曲线E_a（其中$a\in\{0,1\}$）为定义在$\mathbb{F}_2$上的椭圆曲线。定义$\tau(\infty)=\infty$和$\tau(x,y)=(x^2,y^2)$，则 Frobenius 映射$\tau:E_a(\mathbb{F}_{2^m})\to E_a(\mathbb{F}_{2^m})$也是阶为 m 的群同构，并且由于在 $\mathbb{F}_{2^m}$上的点的乘方的便利而使得其计算速度很快。若$P\in E_a(\mathbb{F}_{2^m})$的秩为素数 n 且 n^2不能整除 $\#E_a(\mathbb{F}_{2^m})$，则 $\tau(P)\in\langle P\rangle$，于是 τ 也是$\langle P\rangle$的群同构。令$\mu=(-1)^{1-a}$，由注 3.72 可知方程

$$\lambda^2-\mu\lambda+2\equiv 0 \pmod{n}$$

的两个解 λ 中的一个满足$\tau(P)=\lambda P$。于是τ 是需要的特征，使用 Frobenius 映射下的等价

集时，并行 Pollard's rho 算法的期望运行时间为

$$\frac{1}{M}\sqrt{\frac{\pi n}{2m}}\frac{1}{\theta}。$$

因此可以同时使用 Frobenius 映射和负映射，这时并行 Pollard's rho 算法的期望运行时间为

$$\frac{1}{2M}\sqrt{\frac{\pi n}{m}}+\frac{1}{\theta}。 \tag{4.5}$$

对于 Koblitz 曲线，与式(4.2)相比，速度提高了$\sqrt{2m}$。

例 4.8 （*在互联网上求解 113 位的 ECDLP 实例*）令 E 为素域 $\mathbb{F}_q$ 上的椭圆曲线，$P \in E(\mathbb{F}_q)$ 的阶为素数 n，且 p 和 n 都是 113 位的素数。这样的椭圆曲线的安全性大致与 56 位的 DES 相当。假设我们在互联网上有 $M = 10\,000$ 台计算机可用于求解在这个椭圆曲线$\langle P\rangle$上的 ECDLP 问题的实例，每台计算机可以在 10 微秒内执行一次迭代（算法 4.5 中的步骤 5.2）。若我们选择特殊点的性质使得 $\theta = 2^{-30}$，则算法找到碰撞预计每台计算机所执行的迭代次数约为

$$\frac{\sqrt{\pi 2^{113}}}{2\cdot 10\,000}+2^{30}\approx 9.03\times 10^{13}。$$

于是算法找到碰撞的期望运行时间约为 1045 天或三年。因为特殊点的 x 坐标和(c,d)对可以用 12 个 32 位字存储，所以中心服务器上存储特殊点所需的总空间约为

$$12\theta\frac{\sqrt{\pi n}}{2}\text{个字}\approx 3.8\,\text{GB}。$$

这些结果说明求解 113 位的 ECDLP 问题需要大量的资源，113 位的 ECC 可以为低安全度要求的短期应用提供充分的安全度，但更高安全度的应用需要更长的参数。

多次求解对数问题

在使用并行 Pollard's rho 算法求解$\langle P\rangle$上的一个 ECDLP 问题时，所存储的特殊点可以用来加速求解 $\langle P\rangle$ 上的其他 ECDLP 问题。由于用户共享相同的椭圆曲线参数$E,\mathbb{F}_q$和 P，并选择自己的公钥 $Q \in \langle P\rangle$，因此这个性质关系到椭圆曲线密码系统的安全性。若用 Pollard's rho 算法得到了一个或多个私钥，则求解其他私钥将变得更加容易。

假设$l = \log_P Q$已被计算出来，存储的每个三元组(c,d,X)对应于计算过程中遇到的每个特殊点 X，整数$s = c + dl \bmod n$ 满足 $X = sP$。类似地，整数 $r_j = a_j + b_j l \bmod n$对于

$1 \leqslant j \leqslant L$满足$R_j = r_j P$。现在为了计算$l' = \log_P Q'$，其中$Q' \in \langle P \rangle$，每台处理器从起始点$Y_0 = c_0' P + d_0' Q'$计算序列$Y_i$，其中$c_0', d_0' \in_R [0, n-1]$，并使用同样的迭代函数$f$。对于新序列中遇到的每个特殊点$Y$，满足$Y = c'P + d'Q'$的三元组$(c', d', Y)$被发送到中心服务器。碰撞可能发生在两个新的序列之间，也可能发生在一个新的和一个老的序列之间。对于前者，我们有

$$c'P + d'Q' = c''P + d''Q',$$

其中$l' = (c' - c'')(d'' - d')^{-1} \bmod n$。对于后者，我们有

$$c'P + d'Q' = sP,$$

其中$l' = (s - c')(d')^{-1} \bmod n$。

类似地，在前两个ECDLP计算过程中存储的特殊点可以用来计算第三个ECDLP问题，依次类推。用上述方法迭代求解k个ECDLP问题预计所需要的随机迭代步数为

$$W_k \approx T \sum_{i=0}^{k-1} \frac{\binom{2i}{i}}{4^i},$$

其中T是求解单个ECDLP问题预计所需要的随机迭代步数。因此，求解第二、第三和第四个ECDLP问题需要的时间只是求解第一个ECDLP问题的50%，37%，31%。

我们可以通过选择椭圆曲线的参数使得第一个ECDLP计算上不可行，从而避免后续的ECDLP问题变成更简单的问题。

4.1.3 索引计算攻击

索引计算攻击是在某些群上计算离散对数的最有力方法，如对于有限域上的乘法群$\mathbb{F}_q^*$，定义在有限域$\mathbb{F}_q$上的超椭圆曲线C的雅可比群$J_C(\mathbb{F}_q)$，以及虚二次数域中的类群。于是我们自然要考虑索引计算能否带来求解ECDLP的亚指数时间算法。

我们先概述任意循环群的索引计算，并说明该方法怎样适合求解素域或者二进制域上的乘法群。随后论述对于ECDLP将索引计算用于椭圆曲线群的通常方法为什么不是亚指数时间算法。

索引计算的主要思想

令循环群G的阶为n，α为生成元。假设我们要为$\beta \in G$寻找$\log_\alpha \beta$。索引计算方法的步骤如下：

1. 因数基的选择。选择群 G 的子集 $S=\{p_1, p_2, \ldots, p_t\}$ 作为因数基，使得 G 中的特殊部分元素可以有效地表示为 S 中元素的乘积。S 的选择取决于群 G 的特征。
2. 计算 S 中元素的离散对数。随机选择整数 $k \in [0, n-1]$ 直到 α^k 能表示为 S 中元素的乘积：

$$\alpha^k = \prod_{i=1}^{t} p_i^{c_i}, \text{ 其中 } c_i \geqslant 0。 \tag{4.6}$$

对式(4.6)左右两边以 α 为底取对数得出如下线性等式，其中因数基中元素的对数为未知数：

$$k \equiv \sum_{i=1}^{t} c_i \log_\alpha p_i \pmod{n}。 \tag{4.7}$$

重复这一过程直到得出超过 t 个这样的等式。通过求解线性方程组我们可以得出 $\log_\alpha p_i$，其中 $1 \leqslant i \leqslant t$。

3. 计算 $\log_\alpha \beta$。随机选择整数 k 直到 $\alpha^k \beta$ 能被表示为 S 中元素的乘积：

$$\alpha^k \beta = \prod_{i=1}^{t} p_i^{d_i}, \text{ 其中 } d_i \geqslant 0。 \tag{4.8}$$

对式(4.8)左右两边以 α 为底取对数得出所求的 β 的对数：

$$\log_\alpha \beta = -k + \sum_{i=1}^{t} d_i \log_\alpha p_i \bmod n。 \tag{4.9}$$

索引计算的运行时间极度依赖于因数基 S 的选择。S 的大小 t 的选择也要折中考虑：使用较大的 t 可以使得随机群元素在 S 上成功分解的概率更大；另一方面，选择较小的 t 可以使得需要收集的线性等式的数目更小。t 的优化选择取决于在 S 上进行分解的 G 上元素的性质。

现在考虑素域上乘法群 $G=\mathbb{F}_p^*$ 的情况。$\mathbb{F}_p^*$ 中的元素可以看做是 $[1, p-1]$ 间的整数。有一种自然选择 S 的方法，即选择不大于界限 B 的素数。$\mathbb{F}_p^*$ 中的元素若是 B 光滑的，即它的所有素因子不大于 B，则令其为 S 中的因子。因数基的大小的优化选择取决于 $[1, p-1]$ 间 B 光滑的整数的分布，并且在 $\mathbb{F}_p^*$ 中的 DLP 问题是亚指数算法。该算法最快的方法是数域筛法（NFS），其期望运行时间为 $L_p[\frac{1}{3}, 1.923]$。

再来考虑二进制域上乘法群 $G=\mathbb{F}_{2^m}^*$ 的情况。$\mathbb{F}_{2^m}^*$ 中的元素可以看做是次数小于 m 的非零二元多项式。于是可以选择次数不大于界限 B 的不可约二元多项式作为 S。$\mathbb{F}_{2^m}^*$ 中的元素

若是B光滑的，即它的所有不可约因子的次数不大于B，则令其为 S 中的因子。因数基的大小的优化选择取决于次数不大于B的二元多项式中B光滑的多项式的分布，并且在$\mathbb{F}_{2^m}^*$中的 DLP 问题是亚指数算法。该算法最快的方法是 Coppersmith 算法，对于常数$c < 1.587$，其期望运行时间为$L_{2^m}[\frac{1}{3}, c]$。

索引计算攻击 ECDLP 的失败

假设我们要求解$E(\mathbb{F}_p)$上 ECDLP 的实例，其中$E: y^2 = x^3 + ax + b$是定义在素域$\mathbb{F}_p$上的椭圆曲线。为简单起见，设$E(\mathbb{F}_p)$的阶为素数，这样对于$P \in E(\mathbb{F}_p)$就有$E(\mathbb{F}_p) = \langle P \rangle$。索引计算首先将 E 对应到定义在有理数域$\mathbb{Q}$上的曲线$\widetilde{E}$，即曲线$\widetilde{E}: y^2 = x^3 + \widetilde{a}x + \widetilde{b}$，其中$\widetilde{a}, \widetilde{b} \in \mathbb{Q}$并且$a = \widetilde{a} \bmod p$，$b = \widetilde{b} \bmod p$。于是点$R \in E(\mathbb{F}_p)$对应到点$\widetilde{R} \in \widetilde{E}(\mathbb{Q})$，其坐标模 p 约减后就对应于 R 的坐标。这个对应过程同前文所述用于计算$\mathbb{F}_p^*$和$\mathbb{F}_{2^m}^*$上离散对数的索引计算相似：在$\mathbb{F}_p^*$上的元素对应到$\mathbb{Z}$上的整数，$\mathbb{F}_{2^m}^*$上的元素对应到$\mathbb{F}_2[z]$上的多项式。

著名的 Mordell-Weil 定理规定$\widetilde{E}(\mathbb{Q})$的群结构为$E_{\text{tors}} \times \mathbb{Z}^r$，其中$E_{\text{tors}}$为$\widetilde{E}(\mathbb{Q})$中有限阶的点的集合，$r$ 是非零整数，称为$\widetilde{E}$的秩。此外，Mazur 定理规定E_{tors}较小——事实上$\#E_{\text{tors}} \leqslant 16$。因此选择因数基的直接办法就是选择点集$P_1, P_2, \ldots, P_r$使得$\widetilde{P}_1, \widetilde{P}_2, \ldots, \widetilde{P}_r$在$\widetilde{E}(\mathbb{Q})$中线性无关。式(4.6)的关系的确定可以通过选择对$E(\mathbb{F}_p)$中的点 P 乘以kP，直到对应的$\widetilde{kP}$能够表示为$\widetilde{E}(\mathbb{Q})$中基点的整数线性组合形式：

$$\widetilde{kP} = c_1\widetilde{P}_1 + c_2\widetilde{P}_2 + \cdots + c_r\widetilde{P}_r。$$

于是，对点的坐标模 p 约减满足在$E(\mathbb{F}_p)$中所需的关系：

$$kP = c_1P_1 + c_2P_2 + \cdots + c_rP_r。$$

这里有两个主要的原因使得索引计算方法注定会失败。首先是没有有效的方法将$E(\mathbb{F}_p)$对应到$\widetilde{E}(\mathbb{Q})$。当然对于一个可行的对应方案，对应的点会有小的顶点数（粗略地说，点$\widetilde{P} \in \widetilde{E}(\mathbb{Q})$的顶点数就是$\widetilde{P}$的坐标需要写下的比特数）。然而已证明（在一些合理的假设下）任何椭圆曲线$\widetilde{E}(\mathbb{Q})$中有较小顶点数的点的个数都是非常少的，因此$E(\mathbb{F}_p)$中仅有极少的点能被对应到$\widetilde{E}(\mathbb{Q})$中有较小顶点数的点——这就是索引计算不可避免失败的第二个原因。

对于在非素域$\mathbb{F}_q$上的椭圆曲线 E 的 ECDLP，可以考虑将 E 对应到数域上的椭圆曲线，或者对应到函数域上的椭圆曲线。这些方法由于同样的原因，所以与素域上的情况一样注定会失败。

当然，可能有其他的途径使用索引计算方法来求解 ECDLP。但到目前为止还没有人找到通用的求解 ECDLP 的亚指数（或更快的）算法。

4.1.4 同构攻击

令 E 为定义在有限域 $\mathbb{F}_q$ 上的椭圆曲线，$P \in E(\mathbb{F}_q)$ 的阶为素数 n。令 G 为阶为 n 的群。因为 n 为素数，所以 $\langle P\rangle$ 和 G 都是循环群且同构。若能够有效地计算同构

$$\psi : \langle P\rangle \to G, \tag{4.10}$$

则求解 $\langle P\rangle$ 上 ECDLP 的实例就被有效地约减为求解 G 中 DLP 的实例。即给定 P 和 $Q \in \langle P\rangle$，我们有

$$\log_P Q = \log_{\psi(P)} \psi(Q)。 \tag{4.11}$$

同构攻击将 ECDLP 约减为群 G 中的 DLP，而求解后者已知有亚指数（或更快的）算法。对于ECDLP的攻击者来说，这种方法的特殊之处在于求解特殊类型的椭圆曲线时快于 Pollard's rho 算法。同构攻击的成功特例如下：

(i) 攻击素域异常曲线将定义在素域 $\mathbb{F}_p$ 上的阶为 p 的椭圆曲线上的 ECDLP 约减为加法群 $\mathbb{F}_p^+$ 上模 p 整数的 DLP。

(ii) 当 $\gcd(n, q) = 1$ 时，Weil 和 Tate 配对攻击在 $\langle P\rangle$ 和扩域 $\mathbb{F}_{q^k}$ 上乘法群 $\mathbb{F}_{q^k}^*$ 的阶为 n 的子群间建立同构关系。

(iii) GHS Weil 下降攻击将定义在二进制域 $\mathbb{F}_{2^m}$ 的椭圆曲线中的 ECDLP 约减为 $\mathbb{F}_{2^m}$ 的适当子域上超椭圆曲线雅可比的 DLP。

因为已知有求解 $\mathbb{F}_p^+$ 上 DLP 的多项式时间算法、有限域上乘法群 DLP 和超椭圆曲线雅可比上 DLP 的亚指数时间算法，所以这些情况下的同构攻击对于椭圆曲线密码系统的安全性有严重的影响。下面我们将讨论这些攻击的密码学含义和相应的对策。

攻击素域异常曲线

对于定义在素域 $\mathbb{F}_p$ 上的椭圆曲线 E，若 $\#E(\mathbb{F}_p) = p$，则称为异常曲线。因为阶为素数，所以群 $E(\mathbb{F}_p)$ 为循环群，于是 $E(\mathbb{F}_p)$ 与模 p 整数加法群 $\mathbb{F}_p^+$ 同构。现在 $\mathbb{F}_p^+$ 上的DLP如下：给定 $p, a \in \mathbb{F}_p^+, a \neq 0, b \in \mathbb{F}_p^+$，寻找 $l \in [0, p-1]$ 使得 $la \equiv b \pmod p$。因为 $l = ba^{-1} \bmod p$，所以 $\mathbb{F}_p^+$ 上的 DLP 可以通过使用扩展的 Euclidean 算法（算法 2.20）来计算 $a^{-1} \bmod p$ 而快速地求出。

1997 年，Araki，Satoh，Semaev 和 Smart 提出素域异常椭圆曲线的同构

$$\psi : E(\mathbb{F}_p) \to \mathbb{F}_p^+$$

能够有效地计算。从而该类曲线上的ECDLP能够有效地求解，因此该类椭圆曲线一定不能用于密码协议中。因为判断素域$\mathbb{F}_p$上的一个椭圆曲线E是否为素域异常椭圆曲线的方法很简单（检查是否$\#E(\mathbb{F}_p)=p$即可），所以 Araki-Satoh-Semaev-Smart 攻击在实际使用环境中可以容易地避免。

Weil 和 Tate 配对攻击

假设$P\in E(\mathbb{F}_q)$的素数阶n满足$\gcd(n,q)=1$。令k为满足$q^k\equiv 1 \pmod n$的最小正整数；k为q模n的乘法阶，因此是$n-1$的一个除数。因为n整除q^k-1，扩域$\mathbb{F}_{q^k}$上乘法群$\mathbb{F}_{q^k}^*$有惟一的秩为n的子群G。Weil 配对攻击当附加约束$n\nmid(q-1)$满足时构造从$\langle P\rangle$到G的同构，而 Tate 配对攻击构造$\langle P\rangle$和G的同构而不需要附加的约束。整数k称为嵌入度。

对于大多数椭圆曲线预计有$k\approx n$。这时Weil和Tate配对攻击就不能有效地求解ECDLP，因为有限域$\mathbb{F}_{q^k}$同 ECDLP 参数的大小有指数关系（ECDLP 参数长度为$O(\lg q)$比特，而$\mathbb{F}_{q^k}$的元素长度为$O(k\lg q)$比特）。然而，一些特殊的椭圆曲线的确有很小的嵌入度k。对于这些曲线，Weil 和 Tate 配对约减过程仅需多项式时间。由于已知有$\mathbb{F}_{q^k}$上求解 DLP 的亚指数时间算法，所以在这些特殊椭圆曲线上的 ECDLP 有亚指数时间算法。

有小的嵌入度k的椭圆曲线包括超奇异曲线（定义 3.10）和迹为 2 的椭圆曲线（有$\#E(\mathbb{F}_q)=q-1$）。这些曲线有$k\leqslant 6$，因此不能用于本书中讨论的椭圆曲线密码协议中，除非有限域足够大，以至于$\mathbb{F}_{q^k}^*$上的 DLP 可被认为是难解的。注意，超奇异曲线的一些建设性应用最近已被研究，包括设计基于身份的公钥加密方案（见本章的参考文献）。

为了保证$\mathbb{F}_{q^k}^*$上的椭圆曲线E免于 Weil 和 Tate 配对攻击，只要检查n，即基点$P\in E(\mathbb{F}_q)$的阶，对于所有较小的k不被q^k-1整除就可以了，其中k对应于 DLP 被认为是可解的$\mathbb{F}_{q^k}$。若$n>2^{160}$，则对于所有的$k\in[1,20]$检查上面的条件即可。

Weil 下降

假设E是定义在二进制域$K=\mathbb{F}_{2^m}$上的非超奇异椭圆曲线，设$\#E(\mathbb{F}_{2^m})=nh$，其中n为素数，h较小（如$h=2$或$h=4$）。1998 年，Frey 提出了使用 Weil 下降的方法将$E(\mathbb{F}_{2^m})$上的 ECDLP 约减为K的适当子域$k=\mathbb{F}_{2^l}$上较大亏格曲线雅可比类的 DLP。令$d=m/l$。在 Frey 方法中，使用 Weil 下降攻击方法，首先构造E中标量的所谓 Weil 约束$W_{K/k}$，它是k上的d维交换的变量。接下来试图寻找定义在$W_{K/k}$中k上的曲线C，使得：(i)有比 Pollard's rho 算法更快的求解k上C的雅可比$J_C(k)$的 DLP 的算法；(ii)$E(K)$中的 ECDLP 实例能够有效地映射到$J_C(k)$中的 DLP 实例。

Gaudry，Hess 和 Smart（GHS）给出了如何使 Weil 约束 $W_{K/k}$ 与 $n-1$ 超平面相交，从而从交点中的既约分量最终获得定义在 k 上的亏格 g 的超椭圆曲线 C 的方法。此外，他们还给出了（在大多数情况下）将 $E(K)$ 中的 ECDLP 实例约减为 $J_C(k)$ 中的超椭圆曲线离散对数问题（HCDLP）实例的有效算法。现在对于 $\mathbb{F}_q$ 上的亏格 g 的超椭圆曲线 HCDLP，Enge-Gaudry 索引计算算法有亚指数运行时间 $L_{q^g}[\sqrt{2}]$，位操作为 $g/\log q \to \infty$。于是，假设 g 不是非常大，则 GHS 攻击对于原来的 ECDLP 是亚指数时间算法。

随后证明对于所有密码学意义的所有素数 $m \in [160, 600]$ 的 $\mathbb{F}_{2^m}$ 上的椭圆曲线，GHS 攻击都将失败。注意这些域都只有一个适当的子域，即 $\mathbb{F}_2$。特别地，GHS 攻击生成的超椭圆曲线 C 的亏格或者很小（$J_C(\mathbb{F}_2)$ 太小以至于不能满足 $E(\mathbb{F}_{2^m})$ 中 ECDLP 的非平凡信息），或者太大（$g \geqslant 2^{16}-1$，这时使用已知求解 HCDLP 的方法求解 $J_C(\mathbb{F}_2)$ 的 HCDLP 不可行）。GHS 攻击也被证明对于在特定域 $\mathbb{F}_{2^m}$ 上的所有椭圆曲线也将失败，其中 $m \in [160, 600]$ 为合数；这些域包括 $\mathbb{F}_{2^{169}}$，$\mathbb{F}_{2^{209}}$ 和 $\mathbb{F}_{2^{247}}$。

然而对于 $m \in [160, 600]$ 为合数的 $\mathbb{F}_{2^m}$ 域上的一些椭圆曲线，GHS 攻击对于求解 ECDLP 是有效的。例如，在 $\mathbb{F}_{2^{161}}$ 上的椭圆曲线的 2^{162} 同构类中大约有 2^{94} 类的 ECDLP 可以使用 GHS 攻击，将问题约减为在子域 $\mathbb{F}_{2^{23}}$ 上的亏格 8 的超椭圆曲线的 HCDLP 实例，在约 2^{48} 步解出。因为 Pollard's rho 算法求解 $\mathbb{F}_{2^{161}}$ 上的密码学意义椭圆曲线 ECDLP 需要约 2^{80} 步，所以对于这 2^{94} 类椭圆曲线，GHS 攻击相信会更有效。

令 $\mathbb{F}_{2^m}$ 为二进制域，其中 $m \in [160, 600]$ 为合数，则 GHS 攻击对其中一些可能成功。GHS 攻击能够成功的 $\mathbb{F}_{2^m}$ 上的椭圆曲线的比例相对很小。因此，若随机地在 $\mathbb{F}_{2^m}$ 上选择椭圆曲线，则该曲线抗 GHS 攻击的概率会很高。然而，GHS 攻击失败并不意味着 Weil 下降攻击方法的失败——可能会有其他有用的曲线不是用 GHS 方法构造的，但却满足 Weil 约束，因此为谨慎起见，应该避免使用 m 为合数的 $\mathbb{F}_{2^m}$ 上的椭圆曲线。

4.1.5 相关问题

对于任何椭圆曲线密码方案的安全性，ECDLP 的难解都是必须的，然而仅仅如此还是不够的。我们给出一些与 ECDLP 相关的问题，其难解性在一些椭圆曲线协议安全性证明中作为前提。所有这些问题都能够表示为一般循环群的形式，我们这里则都以椭圆曲线群来进行讨论。

椭圆曲线 Diffie-Hellman 问题

定义 4.9 椭圆曲线 Diffie-Hellman 问题（ECDHP）：给定定义于有限域$\mathbb{F}_q$上的椭圆曲线 E，点 $P \in E(\mathbb{F}_q)$，阶为 n，点 $A = aP, B = bP \in \langle P \rangle$，寻找点$C = abP$。

若$\langle P \rangle$中的 ECDLP 能够有效地解出，首先从(P, A)寻找 a，然后计算$C = aB$，则$\langle P \rangle$中的 ECDHP 也能够有效地解出。因此 ECDHP 不比 ECDLP 困难。还未知 ECDHP 的难度是否等价于 ECDLP，即（假设）给定能够有效求解 ECDHP 的预言器，未知如何有效地求解 ECDLP。然而在一些特殊情况下已证明 ECDLP 和 ECDHP 是等价的，这里 ECDLP 被认为是难解的，例如当 n 为素数且$n-1$的所有素因子都很小时。ECDHP 难解性的最强证明来自于 Boneh 和 Lipton 的结论，他们证明了（在一些合理假设的基础上，假设整数以一定间隔光滑分布）如果 n 为素数且 ECDLP 不能在亚指数时间（对于常数 c）$L_n[\frac{1}{2}, c]$内被求出，那么 ECDHP 就不能在亚指数时间$L_n[\frac{1}{2}, c-2]$内被求出。ECDHP 难解性更进一步的例证可见 Shoup 的在一般素数阶 n 中 Diffie-Hellman 问题的下界$\sqrt{n}$。

椭圆曲线判定性 Diffie-Hellman 问题

ECDHP 是在给定(P, aP, bP)的情况下计算 Diffie-Hellman 秘密点abP。在一些椭圆曲线协议中，出于安全性考虑，需要防止攻击者从abP中获得任何信息。这个要求可以形式化为攻击者不能区分$\langle P \rangle$中的随机元素和abP。

定义 4.10 椭圆曲线判定性 Diffie-Hellman 问题（ECDDHP）：给定定义于有限域$\mathbb{F}_q$上的椭圆曲线 E，点$P \in E(\mathbb{F}_q)$，阶为 n，点$A = aP, B = bP, C = cP \in \langle P \rangle$，判定是否$C = abP$，或者等价地，是否$c \equiv ab \pmod{n}$。

若$\langle P \rangle$中的 ECDHP 能够有效地解出，首先从(P, A, B)中寻找$C' = abP$，然后比较 C 和C'，则$\langle P \rangle$中的 ECDDHP 也能够有效地解出。于是 ECDDHP 不比 ECDHP（ECDLP 同样）困难。关于 ECDDHP 困难性研究的惟一结果是 Shoup 的在一般素数阶 n 中判定性 Diffie-Hellman 问题的下界$\sqrt{n}$。

4.2 参数组

椭圆曲线参数组描述一个椭圆曲线 E 定义的有限域$\mathbb{F}_q$，基点$P \in E(\mathbb{F}_q)$，阶 n。参数的选择要能使得 ECDLP 抵抗所有已知的攻击。这里还会出于安全性或者实现的原因而有其他的约束。典型地，参数组被一组实体共享，而在一些应用中又可能对每个用户特别设

定。在本节中我们将假定使用的是素域（见2.2节）、二进制域（见2.3节）或者最佳扩域（见2.4节）。

定义 4.11 参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$由如下元素构成：

1. 域的阶q。
2. 指示FR（域表示），即域$\mathbb{F}_q$中元素的表示。
3. 种子S，如果椭圆曲线依照算法4.17、算法4.19或算法4.22随机生成。
4. 两个系数$a,b\in\mathbb{F}_q$，定义了$\mathbb{F}_q$上椭圆曲线E的等式（例如在素域或OEF中$y^2=x^3+ax+b$，在二进制域中$y^2+xy=x^3+ax^2+b$）。
5. $\mathbb{F}_q$中的两个域元素x_P和y_P，在仿射坐标中定义了一个有穷远点$P=(x_P,y_P)\in E(\mathbb{F}_q)$。$P$有素数阶，称其为基点。
6. P的阶n。
7. 余因子$h=\#E(\mathbb{F}_q)/n$。

安全性约束 为了避免针对ECDLP的Pohlig-Hellman攻击（见4.1.1节）和Pollard's rho攻击（见4.1.2节），$\#E(\mathbb{F}_q)$必须能被足够大的素数n整除。最小应该保证$n>2^{160}$。给定域$\mathbb{F}_q$，为了最大限度地抵抗Pohlig-Hellman和Pollard's rho攻击，应选择E使得$\#E(\mathbb{F}_q)$为素数或者近似素数，即$\#E(\mathbb{F}_q)=hn$，其中n为素数，h非常小（如$h=1,2,3$或4）。

为了抵抗同构攻击（见4.1.4节），还要有更进一步的预防。为了避免攻击素域异常曲线，应该保证$\#E(\mathbb{F}_q)\neq q$。为了避免Weil和Tate配对攻击，应该保证对于所有的$1\leqslant k\leqslant C$，n不被q^k-1整除，其中C应该足够大，以使$\mathbb{F}_q^*$c上的DLP不可解（若$n>2^{160}$，则$C=20$就已足够）。最后，为了避免Weil下降攻击，应该仅使用m为素数的$\mathbb{F}_{2^m}$。

可验证随机选择椭圆曲线 为了防止针对特殊类型椭圆曲线的攻击，应随机选择满足$\#E(\mathbb{F}_q)$能被大素数整除的椭圆曲线E。因为随机曲线满足已知的同构攻击的条件的概率非常小而可忽略，因此已知攻击仍可避免。可以依照一些预先规定的过程将单向函数例如SHA-1的输出作为定义椭圆曲线的系数，而可证明地随机选择一个曲线。函数的输入种子S作为证据（在假定SHA-1单向性的前提下）证明椭圆曲线的确是随机生成的。这保证了椭圆曲线的用户不被通过故意构造有潜在弱点的曲线而窃取用户的私钥。

4.2.1 参数组的生成和确认

算法4.14单向生成密码学安全的参数组——上面讨论过的所有安全性约束都已满足。参数组可以通过算法4.15明确地确认。确认过程证明正在讨论的椭圆曲线有声明的阶，能

抵抗针对ECDLP的所有已知攻击，且基点有声明的阶。使用不可信的软件或第三方所获得的椭圆曲线同样可以使用该算法来确认其密码学安全性。

参数组的一些例子在 A.2 节中给出。

注 4.12 （*算法 4.14 和算法 4.15 中对于 n 和 L 的约束*）

(i) 因为选择 n 时要满足 $n > 2^L$，所以算法 4.14 的输入要求 $L \geqslant 160$，以确保 $n > 2^{160}$。

(ii) 条件 $L \leqslant \lfloor \mathrm{lb}\, q \rfloor$ 确保 $2^L \leqslant q$，$\mathbb{F}_q$ 上的椭圆曲线 E 的阶 $\#E(\mathbb{F}_q)$ 能被 L 位的素数整除（由于 $\#E(\mathbb{F}_q) \approx q$）。另外，若 $q = 2^m$，则 L 应满足 $L \leqslant \lfloor \mathrm{lb}\, q \rfloor - 1$，因为 $\#E(\mathbb{F}_{2^m})$ 为偶数（见定义 3.18(iii)）。

(iii) 条件 $n > 4\sqrt{q}$ 保证 $E(\mathbb{F}_q)$ 有惟一的阶为 n 的子群，因为由 Hasse 定理（定理 3.7）有 $\#E(\mathbb{F}_q) \leqslant (\sqrt{q}+1)^2$，从而 n^2 不能整除 $\#E(\mathbb{F}_q)$。此外，由于 $hn = \#E(\mathbb{F}_q)$ 取决于 Hasse 区间，从而只有一个可能的 h 使得 $\#E(\mathbb{F}_q) = hn$，即 $h = \lfloor (\sqrt{q}+1)^2/n \rfloor$。

注 4.13 （*选择候选椭圆曲线*）在算法 4.14 中，候选椭圆曲线 E 使用 4.2.2 节中的算法可证明地随机生成。对于素域或 OEF 通过 SEA 点计数算法，对于二进制域通过 Satoh 的点计数算法得到阶 $\#E(\mathbb{F}_q)$。对于 $\mathbb{F}_q$ 上的椭圆曲线 E 的阶 $\#E(\mathbb{F}_q)$，若 $\mathbb{F}_q$ 是素域或 OEF，则该阶粗略地均匀分布于 Hasse 区间 $[q+1-2\sqrt{q}, q+1+2\sqrt{q}]$，若 $\mathbb{F}_q$ 是二进制域，则该阶粗略地均匀分布于 Hasse 区间中的偶数。因此在得到椭圆曲线的素数或近似素数阶以前，可以通过对 Hasse 区间中的每个素数的估计来精确估算椭圆曲线的实际点数。对于候选曲线的测试，可以采用早期中止来加速，首先用 SEA 算法快速确定 $\#E(\mathbb{F}_q)$ 模小素数 l，淘汰 $\#E(\mathbb{F}_q)$ 能被 l 整除的曲线。只有通过这些测试的椭圆曲线才进行全部的点计数算法。

使用随机曲线的一种替代方法是选择曲线的子域或者使用 CM 方法（见 4.2.3 节）。算法 4.14 通过简单的修改后可提供这些选择的方法。

算法 4.14 生成参数组

输入：域的阶为 q，$\mathbb{F}_q$ 的域表示 FR，安全级别 L 满足 $160 \leqslant L \leqslant \lfloor \mathrm{lb}\, q \rfloor$ 和 $2^L \geqslant 4\sqrt{q}$。

输出：参数组 $D = (q, \mathrm{FR}, S, a, b, P, n, h)$。

1. 若 $\mathbb{F}_q$ 是素域、二进制域或 OEF，则对应地用算法 4.17、算法 4.19 或算法 4.22 随机选择 $a, b \in \mathbb{F}_q$。令 S 为生成的种子。若 $\mathbb{F}_q$ 是素域或 OEF，则令 E 为 $y^2 = x^3 + ax + b$；若 $\mathbb{F}_q$ 是二进制域，则 $y^2 + xy = x^3 + ax^2 + b$。
2. 计算 $N = \#E(\mathbb{F}_q)$。见 4.2.3 节。

3. 检验 N 是否能被满足 $n > 2^L$ 的大素数 n 整除。若不能，则跳至步骤 1。
4. 检验 n 对于所有 $1 \leqslant k \leqslant 20$ 是否能被 $q^k - 1$ 整除。若能，则跳至步骤 1。
5. 检验是否 $n \neq q$。若不是，则跳至步骤 1。
6. 令 $h \leftarrow N/n$。
7. 选择任意的点 $P' \in E(\mathbb{F}_q)$ 并令 $P = hP'$。重复直到 $P \neq \infty$。
8. 返回 $(q, \mathrm{FR}, S, a, b, P, n, h)$。

算法 4.15 参数组的确认

输入：参数组 $D = (q, \mathrm{FR}, S, a, b, P, n, h)$。

输出：接受或拒绝参数组 D 的有效性。

1. 检验 q 是否是素数的乘积（$q = p^m$，其中 p 为素数且 $m \geqslant 1$）。
2. 若 $p = 2$，则检验 m 是否为素数。
3. 检验FR是否是有效的域表示。
4. 检验 a, b, x_P, y_P（其中 $P = (x, y)$）是否是 $\mathbb{F}_q$ 上的元素（例如检验它们是否符合 $\mathbb{F}_q$ 上元素正确的格式）。
5. 检验 a 和 b 是否定义 $\mathbb{F}_q$ 上的椭圆曲线（例如对于 $p > 3$ 的域要求 $4a^3 + 27b^2 \neq 0$，对于二进制域要求 $b \neq 0$）。
6. 若椭圆曲线是随机生成的，则
 6.1 检验 S 是长度至少为 l 比特的二进制串，其中 l 为杂凑函数 H 的长度。
 6.2 对素域用算法 4.18，对二进制域用算法 4.21，对 OEF 用算法 4.23，检验 a 和 b 是否由 S 生成。
7. 检验是否 $P \neq \infty$。
8. 检验 P 是否满足由 a 和 b 定义的椭圆曲线方程。
9. 检验 n 是否是素数，$n > 2^{160}$，$n > 4\sqrt{q}$。
10. 检验是否 $nP = \infty$。
11. 计算 $h' = \lfloor (\sqrt{q} + 1)^2 / n \rfloor$，并检验是否 $h = h'$。
12. 检验对于所有 $1 \leqslant k \leqslant 20$，$n$ 是否不被 $q^k - 1$ 整除。
13. 检验是否 $n \neq q$。
14. 若任何一项检验失败，则返回（“无效”）；否则，返回（“有效”）。

4.2.2 可验证随机生成椭圆曲线

算法 4.17、算法 4.19 和算法 4.22 分别给出了素域、二进制域和 OEF 上可验证随机生成椭圆曲线的方法。相应的检验过程在算法 4.18、算法 4.21 和算法 4.23 中分别给出。素域和二进制域的算法源于 ANSI X9.62 标准。

注 4.16 （算法 4.17 和算法 4.22 中参数 r 的解释）假定 $\mathbb{F}_q$ 是特征大于 3 的有限域。若定义在 $\mathbb{F}_q$ 上的椭圆曲线 $E_1: y^2 = x^3 + a_1x + b_1$ 和 $E_2: y^2 = x^3 + a_2x + b_2$ 同构，且满足 $b_1 \neq 0$（于是 $b_2 \neq 0$），则 $a_1^3/b_1^2 = a_2^3/b_2^2$。奇异椭圆曲线 $E: y^2 = x^3 + ax + b$，即在 $\mathbb{F}_q$ 上有 $4a^3 + 27b^2 = 0$ 的椭圆曲线，正好都有 $a = 0$ 和 $b = 0$，或者 $a^3/b^2 = -27/4$。若 $r \in \mathbb{F}_q$ 且 $r \neq 0$ 和 $r \neq -27/4$，则在 $\mathbb{F}_q$ 上正好有两个曲线的同构类 $E: y^2 = x^3 + ax + b$ 且 $a^3/b^2 = r$。于是本质上在算法 4.17 和算法 4.22 的步骤 10 中对于 (a, b) 仅有两种选择。步骤 9 中强加的条件 $r \neq 0$ 和 $r \neq -27/4$ 保证了奇异椭圆曲线被排除在外。最后，我们指出用该方法生成曲线将永远不会得出 $a = 0$ 和 $b \neq 0$ 的椭圆曲线，也永远不会得出 $a \neq 0$ 和 $b = 0$ 的椭圆曲线。这对总体并不产生影响，因为这样的曲线在所有椭圆曲线中仅占微小的部分，从而不会影响算法的随机性。

在素域上随机生成椭圆曲线

算法 4.17 在素域 $\mathbb{F}_p$ 上随机生成椭圆曲线

输入：素数 $p > 3$，l 比特杂凑函数 H。

输出：种子 S，$a, b \in \mathbb{F}_p$ 定义椭圆曲线 $E: y^2 = x^3 + ax + b$。

1. 令 $t \leftarrow \lceil \text{lb } p \rceil$，$s \leftarrow \lfloor (t-1)/l \rfloor$，$v \leftarrow t - sl$。
2. 选择任意的二进制串 S，其长度 $g \geqslant l$ 比特。
3. 计算 $h = H(S)$，令 r_0 为取 h 的最右边 v 比特所得的长度为 v 的二进制串。
4. 令 R_0 为将 r_0 的最左边比特置为 0 所得的二进制串。
5. 令 z 为与二进制表示 S 相等的整数。
6. 对于 i 从 1 到 s，执行
 6.1 令 s_i 为整数 $(z+i) \bmod 2^g$ 的 g 位二进制表示。
 6.2 计算 $R_i = H(s_i)$。
7. 令 $R = R_0 \| R_1 \| \cdots \| R_s$。
8. 令 r 为与二进制表示 R 相等的整数。

9. 若$r=0$或者若$4r+27\equiv 0 \pmod p$，则跳至步骤2。
10. 选择任意的不全为0的$a,b\in\mathbb{F}_p$，使得$r\cdot b^2\equiv a^3 \pmod p$。
11. 返回(S,a,b)。

算法 4.18 检验在素域$\mathbb{F}_p$上随机生成的椭圆曲线

输入：素数$p>3$，l比特杂凑函数H，长度$g\geqslant l$比特的种子S，$a,b\in\mathbb{F}_p$定义椭圆曲线$E:y^2=x^3+ax+b$。

输出：接受或拒绝由算法4.17所生成的E。

1. 令$t\leftarrow\lceil \text{lb}\, p\rceil$，$s\leftarrow\lfloor (t-1)/l\rfloor$，$v\leftarrow t-sl$。
2. 计算$h=H(S)$，令r_0为取h的最右边v比特所得的长度为v的二进制串。
3. 令R_0为将r_0的最左边比特置为0所得的二进制串。
4. 令z为与二进制表示S相等的整数。
5. 对于i从1到s，执行
 5.1 令s_i为整数$(z+i) \bmod 2^g$的g位二进制表示。
 5.2 计算$R_i=H(s_i)$。
6. 令$R=R_0\|R_1\|\cdots\|R_s$。
7. 令r为与二进制表示R相等的整数。
8. 若$r\cdot b^2\equiv a^3 \pmod p$，则返回（“接受”）；否则，返回（“拒绝”）。

在二进制域上随机生成椭圆曲线

算法 4.19 在二进制域$\mathbb{F}_{2^m}$上随机生成椭圆曲线

输入：正整数m，l比特杂凑函数H。

输出：种子S，$a,b\in\mathbb{F}_{2^m}$定义椭圆曲线$E:y^2+xy=x^3+ax^2+b$。

1. 令$s\leftarrow\lfloor (m-1)/l\rfloor$，$v\leftarrow m-sl$。
2. 选择任意的二进制串S，其长度$g\geqslant l$比特。
3. 计算$h=H(S)$，令b_0为取h的最右边v比特所得的长度为v的二进制串。
4. 令z为与二进制表示S相等的整数。
5. 对于i从1到s，执行
 5.1 令s_i为整数$(z+i) \bmod 2^g$的g位二进制表示。
 5.2 计算$b_i=H(s_i)$。
6. 令$b=b_0\|b_1\|\cdots\|b_s$。

7. 若$b=0$，则跳至步骤2。
8. 选择任意的$a \in \mathbb{F}_{2^m}$。
9. 返回(S,a,b)。

注 4.20 （算法 4.19 中 a 的选择）由定理 3.18 (ii)中关于$\mathbb{F}_{2^m}$上的椭圆曲线同构类的论述可知，只要从$\{0,\gamma\}$中选择 a 就足够安全了，其中$\gamma \in \mathbb{F}_{2^m}$且满足 $\mathrm{Tr}(\gamma)=1$。定理 3.18 (iii)表明$\#E(\mathbb{F}_{2^m})$总是偶数，当$a=0$时$\#E(\mathbb{F}_{2^m})$能被 4 整除。

算法 4.21 检验在二进制域$\mathbb{F}_{2^m}$上随机生成的椭圆曲线

输入：正整数m，l比特杂凑函数H，长度$g \geqslant l$比特的种子S，$a,b \in \mathbb{F}_{2^m}$定义椭圆曲线 $E: y^2+xy=x^3+ax^2+b$。

输出：接受或拒绝由算法 4.19 所生成的 E。

1. 令$s \leftarrow \lfloor (m-1)/l \rfloor$，$v \leftarrow m-sl$。
2. 计算$h=H(S)$，令b_0为取h的最右边v比特所得的长度为v的二进制串。
3. 令z为与二进制表示S相等的整数。
4. 对于i从 1 到s，执行
 4.1 令s_i为整数$(z+i) \bmod 2^g$的g位二进制表示。
 4.2 计算$b_i=H(s_i)$。
5. 令$b'=b_0 \| b_1 \| \cdots \| b_s$。
6. 若$b'=b$，则返回（“接受”）；否则，返回（“拒绝”）。

在 OEF 上随机生成椭圆曲线

算法 4.22 在 OEF $\mathbb{F}_{p^m}$上随机生成椭圆曲线

输入：素数$p>3$，次数为m的约减多项式$f(x) \in \mathbb{F}_p[x]$，l比特杂凑函数H。

输出：种子S，$a,b \in \mathbb{F}_{p^m}$定义椭圆曲线$E: y^2=x^3+ax+b$。

1. 令$W \leftarrow \lceil \mathrm{lb}\, p \rceil$，$t \leftarrow W \cdot m$，$s \leftarrow \lfloor (t-1)/l \rfloor$，$v \leftarrow t-sl$。
2. 选择任意的二进制串S，其长度为$g \geqslant l$比特。
3. 计算$h=H(S)$，令T_0为取h的最右边v比特所得的长度为v的二进制串。
4. 令z为与二进制表示S相等的整数。
5. 对于i从 1 到s，执行
 5.1 令s_i为整数$(z+i) \bmod 2^g$的g位二进制表示。

5.2 计算$T_i = H(s_i)$。

6. 令$T_0 \| T_1 \| \cdots \| T_s = R_{m-1} \| \cdots \| R_1 \| R_0$，其中每个$R_i$都是$W$位比特串。
7. 对每个i，$0 \leqslant i \leqslant m-1$，令$r_i = \overline{R}_i \bmod p$，其中$\overline{R}_i$为与二进制表示$R_i$相等的整数。
8. 令r为由p和$f(x)$定义的OEF上的元素$(r_{m-1}, \ldots, r_1, r_0)$。
9. 若在$\mathbb{F}_{p^m}$上有$r = 0$或者$4r + 27 = 0$，则跳至步骤2。
10. 选择任意的不全为0的$a, b \in \mathbb{F}_{p^m}$，使得在$\mathbb{F}_{p^m}$上有$r \cdot b^2 = a^3$。
11. 返回(S, a, b)。

算法4.23 检验在$\mathbb{F}_{p^m}$上随机生成的椭圆曲线

输入：素数$p > 3$，次数为m的约减多项式$f(x) \in \mathbb{F}_p[x]$，l比特杂凑函数H，长度为$g \geqslant l$比特的种子S，$a, b \in \mathbb{F}_{p^m}$定义椭圆曲线$E: y^2 = x^3 + ax + b$。

输出：接受或拒绝由算法4.22所生成的E。

1. 令$W \leftarrow \lceil \text{lb } p \rceil$，$t \leftarrow W \cdot m$，$s \leftarrow \lfloor (t-1)/l \rfloor$，$v \leftarrow t - sl$。
2. 计算$h = H(S)$，令T_0为取h的最右边v比特所得的长度为v的二进制串。
3. 令z为与二进制表示S相等的整数。
4. 对于i从1到s，执行

4.1 令s_i为整数$(z+i) \bmod 2^g$的g位二进制表示。

4.2 计算$T_i = H(s_i)$。

5. 令$T_0 \| T_1 \| \cdots \| T_s = R_{m-1} \| \cdots \| R_1 \| R_0$，其中每个$R_i$都是$W$位比特串。
6. 对每个i，$0 \leqslant i \leqslant m-1$，令$r_i = \overline{R}_i \bmod p$，其中$\overline{R}_i$为与二进制表示$R_i$相等的整数。
7. 令r为由p和$f(x)$定义的OEF上的元素$(r_{m-1}, \ldots, r_1, r_0)$。
8. 若在$\mathbb{F}_{p^m}$上有$r \cdot b^2 = a^3$，则返回（“接受”）；否则，返回（“拒绝”）。

4.2.3 确定椭圆曲线点的个数

在4.2节的简介中已经讨论了在密码协议中使用的椭圆曲线E的阶$\#E(\mathbb{F}_q)$必须满足安全约束。因此在参数组生成过程中要考虑的一个重要因素就是确定椭圆曲线点的个数。最简单的方法是直接对每个$x \in \mathbb{F}_q$通过Weierstrass方程查找$y \in \mathbb{F}_q$的解的个数。这种方法显然不适于密码学用途的域空间很大的运算。在实践中选择已知阶的椭圆曲线通常使用以下三种方法之一。

子域曲线 令$q = p^{ld}$，其中$d > 1$。选择定义在$\mathbb{F}_{p^l}$上的椭圆曲线E，使用直接计数的

方法对 $E(\mathbb{F}_{p^l})$ 上的点计数，然后由定理3.11 可以容易地确定 $\#E(\mathbb{F}_q)$。密码应用中实际使用的是$E(\mathbb{F}_q)$。因为椭圆曲线E定义在$\mathbb{F}_q$的适当子域$\mathbb{F}_{p^l}$上，所以称为子域曲线。例如在 3.4 节中介绍的 Koblitz 曲线就是 $p=2$，$l=1$的子域曲线。因为对于d 的所有约数c，$\#E(\mathbb{F}_{p^{lc}})$ 整除 $\#E(\mathbb{F}_q)$，而我们又需要椭圆曲线的阶为素数或者近似为素数，所以 l 应该较小（最好 $l=1$）而d 应该为素数。

复乘法（CM）方法 该方法首先选择一个满足安全性约束的阶N，然后构造阶为N的椭圆曲线。对于素域上的椭圆曲线，CM方法又称为 Atkin-Morain 方法；对于二进制域称为 Lay-Zimmer 方法。对于阶为 q 的有限域，选择椭圆曲线的秩$N=q+1-t$，使得复乘法域$\mathbb{Q}(\sqrt{t^2-4q})$的类数目很小，这时CM方法是很有效的。在工作站上 1 分钟内就能生成 160 比特域上适合密码学用途的曲线。特别是CM方法比最好的对素域和 OEF 上随机选择的椭圆曲线点的计数算法还要快得多。对于二进制域上的椭圆曲线，已经有比CM方法更快的点的计数算法（见下文）。

因为对于类数目小的椭圆曲线上的 ECDLP 未知有更好的解法，所以使用CM方法生成的椭圆曲线可提供同随机生成的椭圆曲线相同级别的安全性。

点计数 1985 年，Schoof 首先提出了计算任意椭圆曲线E 的 $\#E(\mathbb{F}_q)$ 的多项式时间算法。该算法先对小素数l计算 $\#E(\mathbb{F}_q) \bmod l$，然后用中国剩余定理确定 $\#E(\mathbb{F}_q)$。对于实际密码应用中的 q，该算法实现起来效率很低，但随后该算法被 Atkin 和 Elkies 等人改进，改进后的算法称为 Schoof-Elkies-Atkin（SEA）算法。SEA 算法是已知最好的求解任意素域和 OEF 上椭圆曲线点计数的算法，对于实际密码应用中的 q，该算法的运行时间只需要几分钟。因为它可以快速地确定点数模小素数l的值，所以可用于早期中止策略，快速地将阶能被小素数整除的候选曲线淘汰。

1999 年，Satoh 提出了计算小特征有限域上点计数的全新方法。Satoh 方法的一些变体，如 Satoh-Skjernaa-Taguchi（SST）和算术几何中值（AGM）算法，在二进制域的情况下速度非常快，在工作站上几秒钟内就能生成$\mathbb{F}_{2^{163}}$上适合密码学用途的椭圆曲线。

4.3 密钥对

椭圆曲线密钥对与参数组 $D=(q, \mathrm{FR}, S, a, b, P, n, h)$中的一系列参数相关。在由$P$生成的群 $\langle P\rangle$ 上随机选择点Q作为公钥。相应的私钥是$d=\log_P Q$。要生成密钥对的实体A必须保证参数组是合法的（见 4.2 节）。参数组和公钥间的关系必须能被所有随后可能用到A的

公钥的通信实体所检验。在实践中这种关系可由密码学途径（例如一个认证中心生成证明该关系的证书）或书面（如所有实体使用同样的参数组）达成。

算法 4.24 密钥对的生成

输入：参数组 $D=(q,\mathrm{FR},S,a,b,P,n,h)$。

输出：公钥Q，私钥d。

1. 选择$d\in_R[1,n-1]$。
2. 计算$Q=dP$。
3. 返回(Q,d)。

从公钥Q计算私钥d的问题显然就是椭圆曲线离散对数问题。因此至关重要的是参数组 D 的选择要使得 ECDLP 不可求解。此外数d的生成要是“随机”的，以防止攻击者基于概率搜索策略获得额外信息。

公钥确认

确认公钥的目的是检验公钥拥有可靠的算术性质。执行成功表明对应的私钥逻辑上存在，尽管不能表明某人已经计算出私钥，也不能表明主体确实拥有它。公钥确认在基于 Diffie-Hellman 的密钥建立协议中特别重要，实体A将自己的私钥和从另一个实体B处收到的公钥相结合而得出共享的秘密k，随后使用k进行对称密码协议（如加密或者消息认证）。一个不诚实的B会选择一个非法的公钥通过上述途经使用k窃取A的私钥的信息。

算法 4.25 公钥确认

输入：参数组 $D=(q,\mathrm{FR},S,a,b,P,n,h)$，公钥$Q$。

输出：判断Q是否合法。

1. 检验是否$Q\neq\infty$。
2. 检验x_Q,y_Q是否是$\mathbb{F}_q$上元素的正确表示（例如，若$\mathbb{F}_q$是素域，则应在整数区间$[0,q-1]$，若$\mathbb{F}_q$是阶为2^m的二进制域，则串的长度应为m比特）。
3. 检验Q是否满足由a和b定义的椭圆曲线方程。
4. 检验是否$nQ=\infty$。
5. 若任何检验失败，则返回（“拒绝”）；否则，返回（“接受”）。

检验$nQ=\infty$有比对nQ执行点乘更快的方法。例如当$h=1$（这是实际使用素域上椭圆曲线的常见情况）时，算法 4.25 的步骤 1、步骤 2 和步骤 3 可直接推出$nQ=\infty$。在一些

协议中对$nQ=\infty$的检验可以省略，或者嵌入到协议的计算中，或者由对$hQ\neq\infty$的检验所取代。后者的检验可确保Q不在阶能被h整除的$E(\mathbb{F}_q)$的小子群上。

算法 4.26 嵌入的公钥确认

输入：参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$，公钥$Q$。

输出：判断Q是否（部分）合法。

1. 检验是否$Q\neq\infty$。
2. 检验x_Q,y_Q是否是$\mathbb{F}_q$上元素的正确表示（例如，若$\mathbb{F}_q$是素域，则应在整数区间$[0,q-1]$，若$\mathbb{F}_q$是阶为2^m的二进制域，则串的长度应为m比特）。
3. 检验Q是否在由a和b定义的椭圆曲线上。
4. 若任何检验失败，则返回（“拒绝”）；否则，返回（“接受”）。

小子群攻击

我们通过描述在密码协议中的小子群攻击举例说明检查公钥合法性的重要性，该攻击在没有执行一些检验时会很有效。假设实体A的密钥对(Q,d)对应于参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$。在一次传递椭圆曲线 Diffie-Hellman（ECDH）协议中，另一个拥有D和Q的可信副本的通信实体B选择$r\in_R[1,n-1]$并发送$R=rP$到A。于是A计算点$K=dR$，同时B计算同样的点$K=rQ$。A和B由此得出共享的密钥$k=\mathrm{KDF}(K)$，其中 KDF 为某个密钥生成函数。注意该密钥建立协议仅提供单向认证（A到B），这在一些应用中可能是需要的，例如在被广泛使用的 SSL 协议中服务器对客户端进行认证，而反之则不需要。我们假定A和B随后以消息认证码算法 MAC 使用密钥k来认证相互间的消息。

假设现在A在公钥的确认中省略了对$nQ=\infty$的检验（算法 4.25 中的步骤 4）。令l为余因子h的素因子。在小子群攻击中，B向A发送一个阶为l的点R（而不是一个n阶群$\langle P\rangle$上的点）。A计算$K=dR$以及$k=\mathrm{KDF}(K)$。因为R的阶为l，K的阶也为l（除非$d\equiv 0\pmod l$，这时$K=\infty$）。于是$K=d_lR$，其中$d_l=d \bmod l$。现在当A发送消息m到B时，认证标志为$t=\mathrm{MAC}_k(m)$，B能够重复地选择$l'\in[0,l-1]$直到$t=\mathrm{MAC}_{k'}(m)$，其中$k'=\mathrm{KDF}(K')$，$K'=l'R$——从而$d_l=l'$高概率成立。B攻击成功预计需要尝试的次数为$l/2$。B能够成对使用相关素数阶$l_1,l_2,\ldots,l_s$的不同点R来重复该攻击，并用中国剩余定理将结果合并得到$d \bmod l_1l_2\cdots l_s$。若相关的h较大，则B就能够获得有关A的私钥d的重要信息，并有可能通过穷举搜索推导出所有的d。

在实际使用中h一般很小（如h=1, 2 或 4），这时上面描述的小子群攻击就只能确定d的很少的几个比特。接下来我们介绍一个将小子群攻击扩展为对不同参数组椭圆曲线的攻击。

无效曲线攻击

无效曲线攻击的基础主要是对于定义在$\mathbb{F}_q$上的椭圆曲线E，通常计算点加法的公式没有用到系数b（见 3.1.2 节）。因此，若E'是定义在$\mathbb{F}_q$上的椭圆曲线，其约减的 Weierstrass 方程与E只有系数b不同，则E'和E的加法是相同的。这样的椭圆曲线E'就称为关于E的无效曲线。

假设现在在一次传递 ECDH 协议中A没有对收到的点进行合法性检验。攻击者B选择一个无效曲线E'使得$E'(\mathbb{F}_q)$中包含阶l很小的点R，并将R发送给A。A计算$K = dR$以及$k = \mathrm{KDF}(R)$。使用小子群攻击，当A发送消息m到B时，认证标志为$\mathrm{tag}\ t = \mathrm{MAC}_k(m)$，$B$能确定$d_l = d \bmod l$。通过重复攻击相关素数阶点$R$（在可能的不同无效曲线上），$B$能够最终获得$d$。

防止无效曲线攻击的最简单办法是检查接收到的点是否确实在合法的椭圆曲线上。

4.4 签名方案

数字签名对应于手写签名的数字版本，可用来提供数据源认证、数据完整性和不可否认性。通常由可信的认证中心使用数字签名来签署将通信实体和其公钥相绑定的证书。

定义 4.27 签名方案由四个算法构成：

1. 参数组生成算法生成一组参数组D。
2. 密钥生成算法由输入的一组参数组D生成密钥对(Q, d)。
3. 签名生成算法由输入的一组参数组D、私钥d和消息m来生成签名Σ。
4. 签名验证算法由输入的一组参数组D、公钥Q、消息m和提供的签名Σ来接受或者拒绝签名。

我们假设参数组D有效（见 4.2 节），则公钥Q有效并与D相关联（见 4.3 节）。若Σ确实是签名生成算法由输入(D, d, m)所产生的，则签名验证算法总是接受输入(D, Q, m, Σ)。

接下来的签名方案的安全性概念源于 Goldwasser，Micali 和 Rivest（GMR）。

定义 4.28 一个签名方案，若对于能够使用选择明文攻击的攻击者在计算能力内是不可伪造的，则称为安全的（或 GMR 安全）。换言之，对于一个能够从合法签名者处得到消息签名的攻击者，也不能产生其他新消息（还没有被要求和获得签名）的合法签名。

该安全性定义是很强的——攻击者被赋予了极大的能力（可访问签名预言器），而他的

目标很弱（获得此前没有呈现给签名预言器的任何消息的签名）。有争议说该概念对于一些应用来说过强了——或许攻击者不能获得他们所选择的消息的签名，或者他们能够伪造出签名的消息在实际应用环境是无意义的（也就没有危害）。然而对于数字签名方案的设计者在使用到签名方案的所有环境中，精确地预知分布广泛的攻击者的能力是不可能的。此外，对于设计者来说，阐明通用标准来确定哪个消息将被视为“无意义的”，也是不可能的。因此，基于可能的最强安全性概念来设计签名方案是明智的——GMR 安全正是这样的概念，因而获得认同。

4.4.1 节和 4.4.2 节将介绍两个标准的签名方案，即 ECDSA 和 KCDSA。

4.4.1 ECDSA

椭圆曲线签名算法(ECDSA)是数字签名算法(DSA)的椭圆曲线版本。最广泛标准化的基于椭圆曲线的签名方案包括 ANSI X9.62、FIPS 186-2、IEEE 1363-2000 和 ISO/IEC 15946-2 标准，以及一些标准的草案。

下文中，H表示一个密码杂凑函数，其输出长度不超过 n 比特（若该条件不满足，则可将 H 的输出截断）。

算法 4.29 ECDSA 签名的生成

输入：参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$，私钥$d$，消息$m$。

输出：签名(r,s)。

1. 选择$k\in_R[1,n-1]$。
2. 计算$kP=(x_1,y_1)$并将x_1转换为整数$\bar{x}_1$。
3. 计算$r=\bar{x}_1 \bmod n$。若$r=0$，则跳至步骤 1。
4. 计算$e=H(m)$。
5. 计算$s=k^{-1}(e+dr) \bmod n$。若$s=0$，则跳至步骤 1。
6. 返回(r,s)。

算法 4.30 ECDSA 签名的验证

输入：参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$，公钥$Q$，消息$m$，签名$(r,s)$。

输出：判断签名是否合法。

1. 检验r和s是否是区间$[1,n-1]$内的整数。若任何一个检验失败，则返回（“拒绝该签名”）。

2. 计算$e=H(m)$。
3. 计算$w=s^{-1} \bmod n$。
4. 计算$u_1=ew \bmod n$和$u_2=rw \bmod n$。
5. 计算$X=u_1P+u_2Q$。
6. 若$X=\infty$，则返回（“拒绝该签名”）。
7. 将X的x坐标x_1转换为整数$\bar{x}_1$；计算$v=\bar{x}_1 \bmod n$。
8. 若$v=r$，则返回（“接受该签名”）；否则，返回（“拒绝该签名”）。

签名验证工作的证明 若一条消息m的签名(r,s)确实是由合法的签名者生成的，则$s\equiv k^{-1}(e+dr) \pmod n$。重新整理可得

$$k\equiv s^{-1}(e+dr)\equiv s^{-1}e+s^{-1}rd\equiv we+wrd\equiv u_1+u_2d \pmod n。$$

于是$X=u_1P+u_2Q=(u_1+u_2d)P=kP$，因此$v=r$即为所需。

安全注记

注 4.31（ECDSA 的安全性证明）为了使 ECDSA 是 GMR 安全的，需要$\langle P\rangle$上的 ECDLP 是难解的，并且杂凑函数 H 是密码学安全的（抗原像攻击和抗碰撞）。还没有证明这些条件对于 GMR 安全是否也是足够的。然而在对 H 的合理假设下，ECDSA 已经在一般群模型上被证明是 GMR 安全的（这里用普通群代替群$\langle P\rangle$）。尽管在一般群模型上被证明是安全的，但这并不意味着在使用诸如椭圆曲线群等特殊群的现实世界中是安全的，ECDSA 的安全性在某种程度上是可论证的。

注 4.32（杂凑函数安全性要求的基本原理）若 H 不能抗原像攻击，则敌手 E 可能通过如下方法伪造签名。E 选择任意的整数 l 并计算$Q+lP$的 x 坐标模 n 约减 r。E 令$s=r$并计算$e=rl \bmod n$。若 E 能够找到一条消息 m 使得$e=H(m)$，则(r,s)就是对 m 的合法签名。

若 H 不能抗碰撞，则 E 可能通过如下方法伪造签名。首先找到两条不同的消息 m 和m'使得$H(m)=H(m')$，再要求 A 对 m 签名；签名的结果对m'也是合法的。

注 4.33（在签名的验证中检验 r 和 s 的基本原理）ECDSA 签名的验证过程中步骤 1 检验 r 和 s 是否是区间$[1,n-1]$内的整数。这些检验能被高效地执行，是出于防止对相关的没有进行这些检验的 ElGamal 签名方案的攻击的考虑。下面是不进行$r\neq 0$（更一般地 $r\not\equiv 0 \pmod n$）的检验时，对于 ECDSA 的可能攻击。假设 A 使用素域$\mathbb{F}_p$上的椭圆曲线$y^2=x^3+ax+b$，其中 b 是模 p 的二次剩余，假设 A 使用阶为素数 n 的基点$P=(0,\sqrt{b})$（所有实体可能选择 x 坐标为 0 的基点，以便减小参数组的大小，这看起来是合理的）。敌

手现在能够通过计算$e=H(m)$来伪造 A 的所有消息 m 的签名。这样就可很容易地检查出$(r=0,s=e)$是对 m 的合法签名。

注 4.34 （*每条消息秘密的安全性需求*）在 ECDSA 签名生成中，每条消息秘密 k 与私钥 d 有同样的安全性需求。若敌手 E 知道 A 对一些消息 m 用来生成签名(r,s)的一条消息秘密 k，则 E 能获得 A 的私钥，因为

$$d=r^{-1}(ks-e) \bmod n, \tag{4.12}$$

其中$e=H(m)$（见 ECDSA 签名生成算法中的步骤 5）。此外，Howgrave-Graham 和 Smart 提出，若敌手知道了一些（例如 100 个）签名的消息对应的每条消息秘密的几个（例如 5 个）连续的比特，则敌手就能够容易地算出私钥。这些研究表明每条消息秘密必须安全地生成、安全地存储，并在使用完后安全地销毁。

注 4.35 （*重复使用每条消息秘密*）每条消息秘密 k 应该随机地生成。这保证了每条消息秘密永远不重复，保证这一点很重要，否则私钥 d 就能被算出。假设使用同样的每条消息秘密 k 来生成两条消息m_1和m_2的 ECDSA 签名(r,s_1)和(r,s_2)，则$s_1\equiv k^{-1}(e_1+dr) \pmod n$和$s_2\equiv k^{-1}(e_2+dr) \pmod n$，其中$e_1=H(m_1)$和$e_2=H(m_2)$。于是$ks_1\equiv e_1+dr \pmod n$和$ks_2\equiv e_2+dr \pmod n$。两式相减得$k(s_1-s_2)\equiv e_1-e_2 \pmod n$。若$s_1\not\equiv s_2 \pmod n$，这种情况的概率极大，则

$$k\equiv(s_1-s_2)^{-1}(e_1-e_2) \pmod n。$$

于是敌手就能够确定 k，然后用式(4.12)算出 d。

4.4.2 EC-KCDSA

EC-KCDSA 是韩国基于证书数字签名算法（KCDSA）的椭圆曲线版本。对其的描述源于 ISO/IEC 15946-2 标准。

在下文中，H 表示一个密码杂凑函数，其输出是长度为l_H的比特串。参数组中 n 的二进制长度应该至少为l_H。*hcert* 是签名者的证书数据的杂凑值，证书包括签名者的身份标识、参数组和公钥。签名者的私钥是整数 $d\in_R[1,n]$，公钥是$Q=d^{-1}P$（与本书中其他所有协议中的dP不同）。这使得设计签名的生成和验证过程不需要进行模逆运算。相反，ECDSA 签名的生成和验证分别需要计算$k^{-1} \bmod n$和$s^{-1} \bmod n$。

算法 4.36 EC-KCDSA 签名生成

输入：参数组 $D=(q,\mathrm{FR},S,a,b,P,n,h)$，私钥 d，证书数据的杂凑值 $hcert$，消息 m。

输出：签名(r,s)。

1. 选择$k\in_R[1,n-1]$。
2. 计算$kP=(x_1,y_1)$。
3. 计算$r=H(x_1)$。
4. 计算$e=H(hcert,m)$。
5. 计算$w=r\oplus e$并将w转换为整数$\overline{w}$。
6. 若$\overline{w}\geqslant n$，则$\overline{w}\leftarrow\overline{w}-n$。
7. 计算$s=d(k-\overline{w}) \bmod n$。若$s=0$，则跳至步骤 1。
8. 返回(r,s)。

算法 4.37 EC-KCDSA 签名验证

输入：参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$，公钥$Q$，证书数据的杂凑值 $hcert$，消息m，签名(r,s)。

输出：判断签名是否合法。

1. 检验 r 的长度是否最多是l_H比特，s 是否是区间$[1,n-1]$内的整数。若任何一个检验失败，则返回（“拒绝该签名”）。
2. 计算$e=H(hcert,m)$。
3. 计算$w=r\oplus e$并将w转换为整数$\overline{w}$。
4. 若$\overline{w}\geqslant n$，则$\overline{w}\leftarrow\overline{w}-n$。
5. 计算$X=sQ+\overline{w}P$。
6. 计算$v=H(x_1)$，其中x_1是X的 x坐标。
8. 若$v=r$，则返回（“接受该签名”）；否则，返回（“拒绝该签名”）。

签名验证工作的证明 若一条消息 m 的签名(r,s)确实是由合法的签名者产生的，则 $s\equiv d(k-\overline{w})\ (\mathrm{mod}\ n)$。重新整理可得 $k\equiv sd^{-1}+\overline{w}\ (\mathrm{mod}\ n)$。于是 $X=sQ+\overline{w}P=(sd^{-1}+\overline{w})P=kP$，因此$v=r$即为所需。

注 4.38（使用 $hcert$）在实践中 $hcert$ 定义为签名者的证书数据的杂凑值，应包括签名者的身份标识、参数组和公钥。在杂凑前预先将 $hcert$ 加入消息 m（如在计算$e=H(hcert,m)$时）能够抵抗基于参数组操作的攻击。

注 4.39 （EC-KCDSA 的安全性证明）在假设 S 上离散对数问题是难解的且杂凑函数 H 是随机函数的前提下，对于有限域上乘法群的素数阶子群 S 操作的 KCDSA 已被证明是 GMR 安全的。事实上，若在签名生成过程的步骤 3 和步骤 4 中使用不同的杂凑函数H_1和H_2，则安全性证明的假设更弱：H_1是随机函数，H_2抗碰撞。

在协议的安全性证明中，假设杂凑函数为随机函数是为了满足随机预言模型。该证明并不意味着在杂凑函数并非随机函数的现实世界中该协议是安全的。但是该安全性证明的确给出了协议安全性的保证，除非敌手能够找到杂凑函数区别于随机函数的特点。

若将签名生成算法的步骤 3 中的操作换成$r = H(x_1, y_1)$，则能将 KCDSA 的安全性证明扩展到 EC-KCDSA 上。

4.5 公钥加密方案

公钥加密方案可用来提供机密性。由于比对称密钥算法慢得多，公钥算法更典型地应用在加密少量数据上，例如信用卡号码和PIN，以及用于传输会话密钥，然后由更快的对称密钥算法使用会话密钥来进行大批量的数据加密或消息认证。

定义 4.40 公钥加密方案由四个算法构成：

1. 参数组生成算法生成一组参数组 D。
2. 密钥生成算法由输入的一组参数组 D 生成密钥对(Q, d)。
3. 加密算法由输入的一组参数组 D、公钥 Q 和明文 m 来产生密文 c。
4. 解密算法由输入的一组参数组 D、私钥 d 和密文 c 来拒绝 c 为非法的密文或者产生明文 m。

我们假设参数组 D 有效（见 4.2 节），则公钥 Q 有效并与 D 相关联（见 4.3 节）。若 c 确实是加密算法由输入(D, Q, m)所产生，则签名验证算法总是接受输入(D, d, c)并输出 m。

接下来的公钥加密方案的安全性概念源于 Goldwasser，Micali，Rackoff 和 Simon。

定义 4.41 一个公钥加密方案，如果对于能够使用选择密文攻击的敌手在计算能力内是不能辨别的，则称为安全的。换言之，对于一个敌手，由其选择两个（长度相同的）明文消息m_1和m_2并给定其中之一的密文 c，即使敌手能够获得任意选择密文（不同于目标密文 c）解密出的明文，也不能够判断 c 是由m_1还是m_2加密所得的。

该安全性定义是很强的——敌手尽管自己选择明文消息m_1和m_2并可访问解密预言器，对于判断 c 是由哪个明文加密所得也没有比乱猜更好的办法。公钥加密方案的不可辨别性

对于抵抗选择密文攻击被认为是“正确”的概念而获得认可。

另一个要求的安全特性是，对于一个敌手来说给定一个合法的密文 c 来产生一个不同的合法密文 c'，使得（未知的）明文消息 m 和 m' 相关是不可行的，该安全特性被称为不可扩展性。已经证明一个公钥加密方案对于选择密文攻击的不可辨别性和对于选择密文攻击的不可扩展性是等价的。

4.5.1 ECIES

椭圆曲线加密方案（ECIES）由 Bellare 和 Rogaway 提出，是 ElGamal 公钥加密方案的一种变体。在 ANSI X9.63 和 ISO/IEC 15946-3 标准以及 IEEE P1363a 标准草案中，已使 ECIES 标准化。

在 ECIES 中，使用 Diffie-Hellman 共享秘密来产生两个对称密钥 k_1 和 k_2。密钥 k_1 用于对称密钥密码中加密明文，而 k_2 用于对得出的密文进行认证。认证从直观上能防止选择密文攻击，因为敌手不能自己生成合法的密文。下面列出了使用到的基本密码部件。

1. KDF 是由杂凑函数 H 构成的密钥导出函数。若需要生成一个长度为 l 比特的密钥，则将 KDF(S)定义为杂凑值 $H(S,i)$ 的级联，其中 i 是计数器，当每个杂凑函数得出结果后增一，直到产生 l 比特的杂凑值时止。
2. ENC 是对称密钥加密方案中的加密函数，而 AES、DEC 则是解密函数。
3. MAC 是消息认证码算法，例如 HMAC。

算法 4.42 ECIES 加密

输入：参数组 $D=(q,\mathrm{FR},S,a,b,P,n,h)$，公钥 Q，明文 m。

输出：密文 (R,C,t)。

1. 选择 $k\in_R[1,n-1]$。
2. 计算 $R=kP$ 和 $Z=hkQ$。若 $Z=\infty$，则跳至步骤 1。
3. $(k_1,k_2)\leftarrow \mathrm{KDF}(x_Z,R)$，其中 x_Z 是 Z 的 x 坐标。
4. 计算 $C=\mathrm{ENC}_{k_1}(m)$ 和 $t=\mathrm{MAC}_{k_2}(C)$。
5. 返回 (R,C,t)。

算法 4.43 ECIES 解密

输入：参数组 $D=(q,\mathrm{FR},S,a,b,P,n,h)$，私钥 d，密文 (R,C,t)。

输出：明文 m 或者拒绝该密文。

1. 对 R 进行嵌入的公钥确认（算法 4.26）。若确认失败，则返回（“拒绝该密文”）。

2. 计算$Z=hdR$。若$Z=\infty$，则返回（“拒绝该密文”）。
3. $(k_1,k_2)\leftarrow \mathrm{KDF}(x_Z,R)$，其中$x_Z$是$Z$的$x$坐标。
4. 计算$t'=\mathrm{MAC}_{k_2}(C)$。若$t'\neq t$，则返回（“拒绝该密文”）。
5. 计算$m=\mathrm{DEC}_{k_1}(C)$。
6. 返回(m)。

解密工作的证明 若密文(R,C,t)确实是由合法的加密者加密明文m产生的，则

$$hdR=hd(kP)=hk(dP)=hkQ。$$

于是解密者同加密者计算出同样的密钥(k_1,k_2)，接受该密文并解密出m。

安全注记

注 4.44 （ECIES 的安全性证明）在假设对称密钥密码加密方案和 MAC 算法是安全的，并且在没有具体度量标准（但在合理范围内）计算量下判定 Diffie-Hellman 问题是不可行的前提下，ECIES 被证明是安全的（基于定义 4.41）。这些 Diffie-Hellman 问题也涉及到密钥导出函数 KDF。

注 4.45 （公钥的合法性）共享的秘密点$Z=hdR$是通过将 Diffie-Hellman 共享秘密dkP和h相乘而得到的。这保证了Z是子群$\langle P\rangle$上的点。在解密过程的步骤 2 中，检验$Z\neq\infty$保证了Z的阶正好是n。这与步骤 1 中嵌入的公钥确认一起，能够抵抗在 4.3 节中描述的小子群攻击和无效曲线攻击，在这些攻击中敌手通过发送非法的点R获得有关接收者私钥的信息。

注 4.46 （密钥导出函数的输入）对称密钥k_1和k_2由 Diffie-Hellman 共享秘密Z的x坐标x_Z导出，Z也就是发送者的一次性公钥R。KDF 的输入包含R是必要的，否则方案就是可扩展的，也就不是不可辨别的。敌手能够用相同明文的其他合法密文作为原来的密文，从而很容易地用$-R$将密文(R,C,t)中的R取代。

4.5.2 PSEC

可证明安全曲线加密方案（PSEC）由 Fujisaki 和 Okamoto 提出。这里我们所描述的版本是由密钥封装机制 PSEC-KEM 和 ISO 18033-2 标准草案中描述的数据封装机制 DEM1 相结合而来的。NESSIE 和 CRYPTREC 已经对 PSEC-KEM 进行了评估。

下面列出了使用到的基本密码部件：

1. KDF 是由杂凑函数H构成的密钥导出函数。
2. ENC 是对称密钥加密方案中的加密函数，而 AES 和 DEC 是解密函数。

3. MAC是消息认证码算法，例如HMAC。

算法 4.47 PSEC加密

输入：参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$，公钥$Q$，明文$m$。

输出：密文(R,C,s,t)。

1. 选择$r\in_R\{0,1\}^l$，其中l是n的比特长度。
2. $(k',k_1,k_2)\leftarrow\mathrm{KDF}(r)$，其中$k'$的长度是$l+128$比特。
3. 计算$k=k' \bmod n$。
4. 计算$R=kP$和$Z=kQ$。
5. 计算$s=r\oplus\mathrm{KDF}(R,Z)$。
6. 计算$C=\mathrm{ENC}_{k_1}(m)$和$t=\mathrm{MAC}_{k_2}(C)$。
7. 返回(R,C,s,t)。

算法 4.48 PSEC解密

输入：参数组$D=(q,\mathrm{FR},S,a,b,P,n,h)$，私钥$d$，密文$(R,C,s,t)$。

输出：明文m或者拒绝该密文。

1. 计算$Z=dR$。
2. 计算$r=s\oplus\mathrm{KDF}(R,Z)$。
3. $(k',k_1,k_2)\leftarrow\mathrm{KDF}(r)$，其中$k'$的长度是$l+128$比特。
4. 计算$k=k' \bmod n$。
5. 计算$R'=kP$。
6. 若$R'\neq R$，则返回（“拒绝该密文”）。
7. 计算$t'=\mathrm{MAC}_{k_2}(C)$。若$t'\neq t$，则返回（“拒绝该密文”）。
8. 计算$m=\mathrm{DEC}_{k_1}(C)$。
9. 返回(m)。

解密工作的证明 若密文(R,C,s,t)确实是由合法的加密者加密明文m产生的，则$dR=d(kP)=k(dP)=kQ$。于是解密者同加密者计算出同样的密钥(k',k_1,k_2)，接受该密文并解密出m。

注 4.49（PSEC的安全性证明）在假设对称密钥密码加密方案和MAC算法是安全的，计算Diffie-Hellman问题是不可行的，以及密钥导出函数是随机函数的前提下，PSEC被证明是安全的（基于定义4.41）。

4.6 密钥建立

密钥建立协议的目的是在开放的网络环境中使用共享秘密密钥提供两个或多个实体间的通信。密钥可能用于在对称密钥协议中达成某些密码学意义的目标，例如机密性或数据完整性。

密钥传输协议是一个实体产生密钥并安全地传送给其他实体的密钥建立协议。当明文消息由密钥组成时，ECIES（见 4.5.1 节）可看做是两方的密钥传输协议。密钥协商协议是使用所有参与的实体提供的信息产生共享秘密密钥的密钥建立协议。本节我们将考虑由基本的 Diffie-Hellman 协议得出的两方密钥协商协议。

安全性定义 同由相互熟悉的人们在一个安全的地点会面并通过重复投掷硬币选择出的密钥一样，理想的密钥建立协议应该使得共享秘密密钥有同样的特征。特别是在密码协议中随后使用的密钥应该在任何方式下都不降低协议的安全性。该安全性概念的形式化证明非常困难。我们给出密钥建立协议的非正式的安全特性列表以代替形式化定义。

攻击模型 安全协议应该能抵抗被动攻击和主动攻击。在被动攻击中敌手通过观察少量诚实的实体对协议的执行，来试图阻止协议达到目标。在主动攻击中，敌手通过加入、删除、篡改或重放消息来暗中破坏通信。为了限制提供给密码攻击的可用数据的数量（例如在一个加密应用中使用固定的会话密钥产生出的密文），两个通信实体A和B之间每次执行密钥建立协议都要产生惟一的密钥，称为会话密钥。为了应对已经掌握其他一些会话密钥的敌手，协议应该始终达到这个目标。

基本的安全性目标 密钥建立协议的基本安全性目标是：

1. 隐式密钥认证。若实体 A 被确保除经过特殊身份验证的第二方实体 B 以外其他实体都不能得到会话密钥的任何部分，则一个密钥建立协议就被称为提供了（B 到 A 的）隐式密钥认证。
2. 显式密钥认证。若实体 A 可确保第二方实体 B 能够计算或者已经计算出会话密钥，则一个密钥建立协议就称为提供了（B 到 A 的）密钥确认。若同时提供了（B 到 A 的）隐式密钥认证和确认，则密钥建立协议就称为提供了（B 到 A 的）显式密钥认证。

通信实体双方的显式密钥认证通常需要三次握手（交换消息）。对于一个两方三次握手密钥协商协议，主要的安全性目标是每个实体对于对方的显式密钥认证。

需要的其他安全特性 使用密钥建立协议的具体应用可能需要其他的安全特性。

1. 前向保密。若一个或者多个实体的长期私钥被泄漏，此前由可信的实体建立的会话密钥的秘密性应该不受影响。
2. 密钥泄漏后冒充的防护。假设 A 的长期私钥被泄漏，显然知道该私钥的敌手现在能冒充 A，因为该私钥正好是证明 A 的身份的值。然而可能要求该泄漏不能使敌手对 A 冒充其他实体。
3. 未知密钥共享的防护。实体 A 不能被强制同实体 B 共享一个没有包含 A 的知识的密钥，换言之，A 相信密钥是与某实体 $C \neq B$ 共享的，而 B（正确地）认为密钥是与 A 共享的。

我们给出两个基于椭圆曲线的密钥协商方案，即 4.6.1 节中的 STS 协议和 4.6.2 节中的 ECMQV。这些协议都提供了显式密钥认证，并拥有前向保密、密钥泄漏后冒充的防护以及未知密钥共享的防护等安全特性。

4.6.1 站对站

站对站（STS）协议是 Diffie，van Oorschot 和 Wiener 提出的基于离散对数的密钥协商方案。这里给出如 ANSI X9.63 标准所描述的该方案的椭圆曲线版本。

在下文中，$D = (q, \text{FR}, S, a, b, P, n, h)$ 表示椭圆曲线参数组，KDF 是密钥导出函数（见 4.5.1 节），MAC 是消息认证码算法如 HMAC，SIGN 是带附录的签名方案，如 ECDSA（见 4.4.1 节）或者 RSA 签名方案中的签名生成算法。若协议 4.50 中任何检验失败，则协议运行终止。

协议 4.50 站对站密钥协商

目标：A 和 B 建立共享秘密密钥。

协议消息：

$$
\begin{aligned}
A \rightarrow B:&\quad A,\ R_A \\
A \leftarrow B:&\quad B,\ R_B,\ s_B = \text{SIGN}_B(R_B, R_A, A),\ t_B = \text{MAC}_{k_1}(R_B, R_A, A) \\
A \rightarrow B:&\quad s_A = \text{SIGN}_A(R_A, R_B, B),\ t_A = \text{MAC}_{k_1}(R_A, R_B, B)
\end{aligned}
$$

1. A 选择 $k_A \in_R [1, n-1]$，计算 $R_A = k_A P$，并发送 A, R_A 到 B。
2. B 执行如下操作：
 2.1 对 R_A 执行嵌入的公钥确认（算法 4.26）。
 2.2 选择 $k_B \in_R [1, n-1]$ 并计算 $R_B = k_B P$。

2.3 计算$Z=hk_BR_A$并检验$Z\neq\infty$。

2.4 $(k_1,k_2)\leftarrow\mathrm{KDF}(x_Z)$，其中$x_Z$是$Z$的$x$坐标。

2.5 计算$s_B=\mathrm{SIGN}_B(R_B,R_A,A)$和$t_B=\mathrm{MAC}_{k_1}(R_B,R_A,A)$。

2.6 发送B,R_B,s_B,t_B到A。

3. A执行如下操作：

3.1 对R_B执行嵌入的公钥确认（算法4.26）。

3.2 计算$Z=hk_AR_B$并检验$Z\neq\infty$。

3.3 $(k_1,k_2)\leftarrow\mathrm{KDF}(x_Z)$，其中$x_Z$是$Z$的$x$坐标。

3.4 检验s_B是B对消息(R_B,R_A,A)的签名。

3.5 计算$t=\mathrm{MAC}_{k_1}(R_B,R_A,A)$并检验$t=t_B$。

3.6 计算$s_A=\mathrm{SIGN}_A(R_A,R_B,B)$和$t_A=\mathrm{MAC}_{k_1}(R_A,R_B,B)$。

3.7 发送s_A,t_A到B。

4. B执行如下操作：

4.1 检验s_A是A对消息(R_A,R_B,B)的签名。

4.2 计算$t=\mathrm{MAC}_{k_1}(R_A,R_B,B)$并检验$t=t_A$。

5. 会话密钥是k_2。

共享的秘密是$Z=hk_Ak_BP$，它由临时的（一次性）公钥R_A和R_B产生。乘以h和检验$Z\neq\infty$可保证Z的阶为n并在群$\langle P\rangle$中。签名$s_A=\mathrm{SIGN}_A(R_A,R_B,B)$和$s_B=\mathrm{SIGN}_B(R_B,R_A,A)$的检验成功可确保每个实体相信对方的身份（因为做签名的实体的身份可由它的签名公钥来确认）、通信未被篡改（假设签名方案是安全的），以及对方知道正在与它通信的实体的身份（因为在被签名的消息中包含了身份信息）。认证目标t_A和t_B的检验成功使每个实体相信对方确实计算出了共享的秘密Z（因为计算认证目标需要关于k_1的知识，从而要知道Z）。

4.6.2 ECMQV

ECMQV是在ANSI X9.63、IEEE 1363-2000和ISO/IEC 15946-3中标准化的三次握手密钥协商协议。

在下文中，$D=(q,\mathrm{FR},S,a,b,P,n,h)$表示椭圆曲线参数组，$(Q_A,d_A)$是$A$的密钥对，$(Q_B,d_B)$是$B$的密钥对，KDF是密钥导出函数（见4.5.1节），MAC是消息认证码算法如HMAC。若R是椭圆曲线上的点，则将$\overline{R}$定义为整数$(\bar{x}\bmod 2^{\lceil f/2\rceil})+2^{\lceil f/2\rceil}$，其中$\bar{x}$是$R$的$x$坐标的整数表示，$f=\lfloor\mathrm{lb}\,n\rfloor+1$是$n$的二进制长度。若协议4.51中任何检验失败，则协议运行终止。

协议 4.51 ECMQV 密钥协商

目标：A 和 B 建立共享秘密密钥。

协议消息：

$A \rightarrow B$: A, R_A

$A \leftarrow B$: B, R_B, $t_B = \mathrm{MAC}_{k_1}(2, B, A, R_B, R_A)$

$A \rightarrow B$: $t_A = \mathrm{MAC}_{k_1}(3, A, B, R_A, R_B)$

1. A 选择 $k_A \in_R [1, n-1]$，计算 $R_A = k_A P$，并发送 A, R_A 到 B。
2. B 执行如下操作：
 2.1 对 R_A 执行嵌入的公钥确认（算法 4.26）。
 2.2 选择 $k_B \in_R [1, n-1]$ 并计算 $R_B = k_B P$。
 2.3 计算 $s_B = (k_B + \overline{R}_B d_B) \bmod n$ 和 $Z = hs_B(R_A + \overline{R}_A Q_A)$，并检验 $Z \neq \infty$。
 2.4 $(k_1, k_2) \leftarrow \mathrm{KDF}(x_Z)$，其中 x_Z 是 Z 的 x 坐标。
 2.5 计算 $t_B = \mathrm{MAC}_{k_1}(2, B, A, R_B, R_A)$。
 2.6 发送 B, R_B, t_B 到 A。
3. A 执行如下操作：
 3.1 对 R_B 执行嵌入的公钥确认（算法 4.26）。
 3.2 计算 $s_A = (k_A + \overline{R}_A d_A) \bmod n$ 和 $Z = hs_A(R_B + \overline{R}_B Q_B)$，并检验 $Z \neq \infty$。
 3.3 $(k_1, k_2) \leftarrow \mathrm{KDF}(x_Z)$，其中 x_Z 是 Z 的 x 坐标。
 3.4 计算 $t = \mathrm{MAC}_{k_1}(2, B, A, R_B, R_A)$ 并检验 $t = t_B$。
 3.5 计算 $t_A = \mathrm{MAC}_{k_1}(3, A, B, R_A, R_B)$ 并发送 t_A 到 B。
4. B 计算 $t = \mathrm{MAC}_{k_1}(3, A, B, R_A, R_B)$ 并检验 $t = t_A$。
5. 会话密钥是 k_2。

协议 4.51 可以看做是普通 Diffie-Hellman 密钥协商协议的扩展版本。值

$$s_A = (k_A + \overline{R}_A d_A) \bmod n$$

作为 A 的临时公钥 R_A 的隐式签名。其中，“签名”的意思是惟一能够计算 s_A 的人是 A，“隐式”的意思是在产生共享秘密 Z 时 B 使用 $s_A P = R_A + \overline{R}_A Q_A$ 间接地检验其合法性。同样地，s_B 是 B 的临时公钥 R_B 的隐式签名。共享秘密是 $Z = hs_A s_B P$，而不是普通 Diffie-Hellman 中的 $k_A k_B P$，乘以 h 和检验 $Z \neq \infty$ 可保证 Z 的阶为 n 并在群 $\langle P \rangle$ 中。注意，Z 与通信双方的长期公钥（Q_A 和 Q_B）一样由临时公钥（R_A 和 R_B）产生。在 MAC 的输入中包含字符串“2”和“3”，是为了区别分别由发起者 A 和接收者 B 产生的认证目标 t_A 和 t_B。认证目

标t_A和t_B的检验成功可使每个实体相信对方确实计算出了共享的秘密Z（因为计算认证目标需要关于k_1的知识，从而要知道Z）、通信未被篡改（假设 MAC 是安全的），以及对方知道正在与它通信的实体的身份（因为在消息认证码中包含了身份信息）。还没有对于协议 4.51 的形式化证明。

4.7 注释和进一步参考

4.1 节

离散对数问题下界的普通群模型由 Nechaev[344]和 Shoup[425]提出。Pohlig-Hellman 算法由 Pohlig 和 Hellman[376]提出。

尽管在传统计算机上ECDLP看起来是难解的，但在量子计算机（使用量子原理机制的计算设施）上求解 ECDLP 则是容易的。1994 年，Shor[424]提出了在量子计算机上计算离散对数和因数分解的多项式时间算法。对于 ECDLP 的情况，Proos 和 Zalka[384]做了更深入的研究，他们设计了执行椭圆曲线群运算的量子电路。Proos 和 Zalka 指出在 K-qubit 量子计算机上能够很容易地求解 k 比特的 ECDLP 实例，其中$K \approx 5k + 8\sqrt{k} + 5 \operatorname{lb} k$（qubit 是量子计算中对应于传统二进制位的基本单位）。而 Beauregard[31]指出在 K-qubit 量子计算机上能够很容易地分解 k 比特的整数，其中$K \approx 2k$。例如在传统计算机上求解 256 位的 ECDLP 实例大致相当于 3072 位的因数分解问题的实例。然而前者可在 1448-qubit 量子计算机上求解，而后者需要 6144-qubit 量子计算机。因此求解因数分解问题比在传统计算机上大致相等难度的求解 ECDLP 问题实例需要更大规模（某种迹象表明这更难构造）的量子计算机。于是引人注意的是人们什么时间造出或者能否造出大规模的量子计算机。这是一个非常活跃并充满猜测的研究领域。到目前为止，最重大的实验结果是 2001 年使用 Vandersypen 等[464]建造的 7-qubit 量子计算机来以 Shor 算法对整数 15 进行因数分解。类似的实验能否大规模用于密码学意义的因数分解和求解 ECDLP 实例，还有待进一步研究。Nielsen 和 Chuang[347]的著作是一本极好的全面概览量子计算领域的图书。

Flajolet 和 Odlyzko[143]研究了随机函数的特征，包括由随机函数产生的序列预计的尾长度和循环长度。计算离散对数的 rho 算法（算法 4.3）由 Pollard[379]提出。Pollard 的原始算法使用三个分支的迭代函数。Teske[458]提供的实验结果表明 Pollard 的迭代函数没有最理想的随机特征，并提出了算法 4.3 中的迭代函数。Teske[458，459]给出了理论和实验证明她的迭代函数当分支数$L = 20$时很接近随机函数模型。

通过使用 Brent 的循环查找算法[70]取代 Floyd 算法，能够加速 Pollard's rho 算法。这使得预计的群操作数从 $3\sqrt{n}$ 减少为约 $2\sqrt{n}$。Sedgewick，Szymanski 和 Yao[419]提出了速度更快的算法，但需要相当的存储空间。

Pollard's rho 算法的并行化版本（算法 4.5）由 van Oorschot 和 Wiener[463]提出。

Gallant，Lambert 和 Vanstone[159]以及 Wiener 和 Zuccherato[482]分别独立地发现了使用自同构加速（并行）Pollard's rho 算法的方法。他们也描述了如何检查一个处理器进入短（且无用的）循环的技术。他们的方法由 Duursma，Gaudry 和 Morain[128]推广到了超椭圆曲线和其他曲线上。

Silverman 和 Stapleton[434]首先研究了在用 Pollard's rho 算法求解 ECDLP 实例时不同点的碰撞能够用于随后的 ECDLP 实例（有相同的椭圆曲线参数）的求解中。Kuhn 和 Struik[271]分析了使用 Pollard's rho 算法来重复求解多个 ECDLP 实例。Kuhn 和 Struik 还证明了给定若干 ECDLP 实例求解任何 k 的最好的策略是任意选择这些实例并投入全部精力对其求解。

Pollard 的袋鼠算法[379]（介绍中也称为 lambda 算法），设计用来在长度为 b 的已知区间内寻找离散对数。其期望的运行时间为 $3.28\sqrt{b}$ 次群运算，需要的存储空间近于零。Van Oorschot 和 Wiener[463]给出了存储需求和期望运行时间的折中改进方法，期望的运行时间为 $2\sqrt{b}$ 次群运算。他们也展示了如何将袋鼠方法并行化，达到同使用的处理器数量呈线性关系的加速比。当没有预先关于离散对数（如 $b=n$）的信息时，并行袋鼠方法比并行 rho 算法更慢，当 $b<0.39n$ 时则更快。Pollard[381]更进一步分析了袋鼠方法。

对于 ECDLP 索引计算失败的论点由 Miller[325]提出，Silverman 和 Suzuki[432]进一步详细阐述了该论点。该方法和其他攻击失败的详细展示，可参见 Koblitz[257]。

Silverman[431]提出了一种对 ECDLP 的攻击，他称之为 xedni 计算。给定定义于素域 $\mathbb{F}_p$ 上的椭圆曲线 E 的 ECDLP 实例 (P, Q)，首先将 P 和 Q 进行 $r \leqslant 9$ 种不同的整数线性组合，并将这 r 个点映射到有理平面 $\mathbb{Q}\times\mathbb{Q}$ 上的点。接下来试图寻找通过这些点的定义在 $\mathbb{Q}$ 上的椭圆曲线 $\widetilde{E}$（这个过程与索引计算相反，索引计算首先映射曲线然后找点，因此称为“xedni”）。若 $\widetilde{E}(\mathbb{Q})$ 的秩小于 r，则 r 个点之间有整数线性相关，于是（几乎确定）得出原来 ECDLP 的解。为了增加 $\widetilde{E}(\mathbb{Q})$ 的秩小于 r 的概率，Silverman 指出要选择 $\widetilde{E}$ 使得 $\#\widetilde{E}(\mathbb{F}_t)$ 对所有小素数 t 都尽可能地小，也就是 $\#E(\mathbb{F}_t)\approx t+1-2\sqrt{t}$（相反的条件 $\#E(\mathbb{F}_t)\approx t+1+2\sqrt{t}$ 称为 Mestre 条件，它由 Mestre[324]提出并成功用于获得 $\mathbb{Q}$ 上高于预计秩的椭圆曲线）。在 Silverman 提出 xedni 计算不久，Koblitz（见[431]的附录 K）发现 xedni 能够改造为求解原始的离散对数问题和因数分解问题。因此，若 xedni 计算攻击有效，则它将对所有重要的公钥方案的安

全性带来负面影响。(对于公钥密码的支持者)幸运的是，Jacobson，Koblitz，Silverman，Stein 和 Teske[222]证明了 xedni 计算是趋于无效的(当$p \to \infty$)，并提供了令人信服的实验证明对于密码学应用规模的素数 p，xedni 计算是非常无效的。

对素域异常椭圆曲线的同构攻击分别由 Satoh 和 Araki[401]，Semaev[420]以及 Smart[438]独立地提出。Semaev 的攻击方法被 Rück[397]归纳为定义在特征 p 的有限域上任意曲线(包括超椭圆曲线)雅可比的阶为 p 的子群上的 DLP。

Weil 配对攻击由 Menezes，Okamoto 和 Vanstone[314]提出，Tate 配对攻击由 Frey 和 Rück[150]提出。Balasubramanian 和 Koblitz[27]证明了对于定义在素域上的大多数椭圆曲线嵌入度 k 都很大。Tate 配对攻击用于有限域 $\mathbb{F}_q$(满足如下条件，即基本元素的阶满足 $\gcd(n, q) = 1$)上任何非奇异不可约曲线的雅可比。Galbraith[155]对于有限域上 g 维超奇异阿贝耳族给出了嵌入度的上界 $k(g)$；这些族包括亏格 g 的超奇异曲线的雅可比。Rubin 和 Silverberg[396]改进了该界限。超奇异曲线(和通常的双线性映射)的应用包括 Joux[227]的三方一次 Diffie-Hellman 协议，Boneh 和 Franklin[58，59]的基于身份的公钥加密方案，Horwitz 和 Lynn[199]以及 Gentry 和 Silverberg[170]的基于分级身份的加密和签名方案，Boneh，Lynn 和 Shacham[62]的短签名方案，Boneh，Gentry，Lynn 和 Shacham[60]的聚集签名方案，Verheul[472]的盲证书方案，Boneh，Mironov 和 Shoup[63]的可证明安全的高效签名方案。

Frey 在 ECC'98 会议(见[149])上的演讲中首先提出 Weil 下降攻击方法。Galbraith 和 Smart[158]进一步阐述了 Frey 的思想(也可见 Hess[196])。Menezes 和 Qu[315]指出该方法对于密码学意义而言的在所有素数 $m \in [160, 600]$ 的 $\mathbb{F}_{2^m}$ 上的所有椭圆曲线都将失败。Jacobson，Menezes 和 Stein[223]使用 GHS 攻击解决了实际的 $\mathbb{F}_{2^{124}}$ 上的 ECDLP 实例，他们首先将其约减为 $\mathbb{F}_{2^4}$ 上的亏格31的超椭圆曲线 HCDLP 实例，并对后者使用 Enge-Gaudry 亚指数时间算法[163，133]进行求解。Maurer，Menezes 和 Teske[304]通过对于最易受 GHS 攻击的合数 $m \in [160, 600]$ 的 $\mathbb{F}_{2^m}$ 上的椭圆曲线同构类的辨别和枚举，完成了对 GHS 攻击的分析。Menezes，Teske 和 Weng[318]指出对于能被 5 整除的 $m \in [185, 600]$ 的域 $\mathbb{F}_{2^m}$ 作为椭圆曲线密码偏弱，因为对于这些域上的所有密码学意义的椭圆曲线来说，用 GHS 攻击来求解 ECDLP 比 Pollard's rho 算法明显更快。

对于定义在 $\mathbb{F}_{q^n}$ 上的椭圆曲线 E_1 和 E_2，若 $\#E_1(\mathbb{F}_{q^n}) = \#E_2(\mathbb{F}_{q^n})$，则称为在 $\mathbb{F}_{q^n}$ 上同源。Galbraith，Hess 和 Smart[156]给出了在 $\mathbb{F}_{q^n}$ 上两个同源椭圆曲线明确计算同源的实现算法。他们研究发现他们的算法能够用于通过如下方法提高 GHS 攻击的效率。给定 $\mathbb{F}_{2^m}$ 上某密码学意义椭圆曲线 E_1 的 ECDLP 实例，检查 E_1 是否和 E_2 在 $\mathbb{F}_{2^m}$ 上同源，而 E_2 上 GHS 能约减为比 E_1 更简单的 HCDLP 实例。然后使用同源 $\phi : E_1 \to E_2$ 将 ECDLP 实例映射到 $E_2(\mathbb{F}_{2^m})$，并

使用 GHS 攻击对后者进行求解。例如当 $m = 155$时，估计在$\mathbb{F}_{2^{155}}$上的椭圆曲线的2^{156}个同构类中大约有2^{104}个能同源映射到约2^{32}个$\mathbb{F}_{2^{155}}$上被认为易受 GHS 攻击的椭圆曲线。于是 GHS 攻击就可用于$\mathbb{F}_{2^{155}}$上2^{156}个椭圆曲线中的2^{104}个。

Arita[18]指出在特征 3 的有限域$\mathbb{F}_{3^m}$上的一些椭圆曲线也易受 Weil 下降攻击。Diem [118, 119]指出 GHS 攻击可扩展到 $p \geqslant 5$为素数的$\mathbb{F}_{p^m}$上的椭圆曲线上。他总结出当 m 为素数且$m \geqslant 11$时，他提出的 GHS 攻击变体总会失败——这是因为在高亏格曲线上的离散对数问题也是难解的。然而他给出的一些证据证实了当 $m = 3, 5, 7$时对于某些椭圆曲线该攻击可能会成功。要对Diem的研究工作对于密码学蕴含的意义做全面理解，需要更深入的研究和实践。

Den Boer[112]证明了在任意阶为 n 的循环群上离散对数和 Diffie-Hellman 问题是等价的，其中 n 满足$\phi(n)$没有大素数因子（$\phi(n)$是欧拉函数）。Maurer[305]对其结果进行了推广；见 Maurer 和 Wolf[307]。Boneh 和 Lipton[61]形式化了普通域（他们称之为黑盒域）上的问题，得出了 ECDLP 困难性包含 ECDHP 困难性的证明。Boneh 和 Shparlinski[64]证明了若定义在素域$\mathbb{F}_p$上的椭圆曲线 E 的素数阶子群$\langle P \rangle \subseteq E(\mathbb{F}_p)$上的 ECDHP 是难解的，则对于 E 的大多数同构椭圆曲线不存在预知 Diffie-Hellman 秘密点的 x 坐标或 y 坐标的最低位的有效算法。这并不排除对于每个椭圆曲线同构的有效预知算法的存在。Boneh 和 Shparlinski 的工作表明计算从(P, aP, bP)得出的 Diffie-Hellman 秘密点abP的 x 坐标或 y 坐标的最低位的难度，同计算整个点abP的难度相当。

Boneh[54]给出了判定性 Diffie-Hellman 问题及其密码学应用的全面概览（circa 1998）。Joux 和 Nguyen[229]（见 Verheul[471]）给出了离散对数问题和 Diffie-Hellman 问题是等价的（并且未知在多项式时间内可解）超奇异椭圆曲线的示例，但上面的判定性 Diffie-Hellman 问题是可解的。

4.2 节

算法 4.14 和算法 4.15（参数组的生成和确认），以及算法 4.17、算法 4.18、算法 4.19 和算法 4.21（素域和二进制域上随机椭圆曲线的生成和检验）摘自 ANSI X9.62[14]。Vaudenay[467]研究了随机椭圆曲线的生成过程，并给出了一些改进。特别是他提出了域的阶和二进制域情况下输入的表示。

Lenstra[285]证明了素域上的椭圆曲线的阶大致均匀分布在 Hasse 区间上。Howe[201]扩展了 Lenstra 的结果，对于任何有限域$\mathbb{F}_q$和素数幂 l^k，得到了在$\mathbb{F}_q$上随机选择的椭圆曲线的阶$\#E(\mathbb{F}_q)$能被 l^k整除的概率。Lercier[287]首先研究了早期中止策略。

Atkin 和 Morain[20]（见 Buchmann 和 Baier[79]）对于素域，Lay 和 Zimmer[276]对于二进制域，Baier 和 Buchmann[24]对于最佳扩域，分别给出了复乘法方法。Weng[479]提出了用 CM 方法生成适于密码应用的亏格 2 超椭圆曲线。

Schoof 算法[411]最初描述了奇数特征有限域上的椭圆曲线情况，后来被 Koblitz[252]改变为二进制域的情况。Blake，Seroussi 和 Smart[49，第 VII 章]对于 Schoof 算法[411]及其由 Atkin 和 Elkies（和其他人）完成的改进进行了更深入的研究。Lercier 和 Morain[289]以及 Izu，Kogure，Noro 和 Yokoyama[218]报告了他们在素域情况下的 SEA 算法的实现。后者于 300 MHz Pentium II 上在约 7.5 分钟内实现了 240 比特素域上的点的计数，并能够在约 3 小时内生成 240 比特素域上阶为素数的椭圆曲线。Gaudry 和 Harley[166]研究了将 Schoof 算法扩展到亏格 2 超椭圆曲线的方法。

Satoh[400]给出了大于 5 的小特征有限域上椭圆曲线的点计数算法。Fouquet，Gaudry 和 Harley[146]及 Skjernaa[436]将其扩展到了二进制域的椭圆曲线上。随后提出了很多二进制域情况下的变体。其中 Vercauteren，Preneel 和 Vandewalle[470]提出了低存储需求的变体方法。Fouquet，Gaudry 和 Harley[147]探究了与早期中止策略的结合，目的是生成近似素数阶的椭圆曲线。Satoh，Skjernaa 和 Taguchi[402]提出了 SST 变体方法。AGM 方法由 Mestre，Harley 和 Gaudry 提出，Gaudry[164]对其进行了描述并给出了改进，并与 AGM 算法和 SST 算法进行了比较。Gaudry 称其修改后的 SST 算法能够在 700 MHz Pentium III 上分别于 0.13 秒和 0.40 秒内确定 $\mathbb{F}_{2^{163}}$ 和 $\mathbb{F}_{2^{239}}$ 上随机选择的椭圆曲线点的计数。Kim 等[243]，Lercier 和 Lubicz[288]及 Harley[192]对于二进制域上小型高斯标准基的情况给出了进一步的改进。

另一个值得注目的算法是 Kedlaya[240]提出的在小奇数特征有限域上超椭圆曲线（任意亏格）的点的计数算法。Kedlaya 的算法被 Vercauteren[469]扩展到了二进制域的超椭圆曲线上，Gaudry 和 Gürel[165]将其扩展到了不同于 r 的小特征有限域上的超奇异椭圆曲线 $y^r = f(x)$ 上，Denef 和 Vercauteren[113]将其扩展到了二进制域上的 Artin-Schreier 曲线 $y^2 + x^m y = f(x)$ 上。

4.3 节

公钥确认的合法性由 Johnson[224，225]在各种标准制定会议上传播。针对离散对数的小子群攻击由 Vanstone（Menezes，Qu 和 Vanstone [316]），van Oorschot 和 Wiener [462]，Anderson 和 Vaudenay [13]以及 Lim 和 Lee [296]提出。无效曲线攻击是小子群攻击对无效曲线的扩展，使用的思想基于 Biehl，Meyer 和 Müller[46]提出的对于椭圆曲线方案的差分故障攻击。无效曲线攻击首先由 Antipa，Brown，Menezes，Struik 和 Vanstone[16]提出，他们

还展示了对于 ECIES 加密方案和一次 ECMQV 密钥协商协议的潜在有效性。

4.4 节

签名方案的思想由 Diffie 和 Hellman[121]于 1976 年提出。ElGamal[131]于 1984 年提出了首个基于离散对数问题的签名方案。ElGamal 方案的变体有很多，包括 DSA，KCDSA 以及 Schnorr [410]和 Okamoto[354]提出的方案。GMR 安全性（定义 4.28）的概念由 Goldwasser，Micali 和 Rivest[175]提出。

ECDSA 由 Johnson，Menezes 和 Vanstone[226]设计。Brown[75]对其进行了深入的安全性分析，证明了 ECDSA 在普通群模型上的 GMR 安全性。Dent[114]指出在普通群模型上的安全性证明并不能确保在实践中一个签名方案的安全性，因为该方案可能在普通群模型上能证明是安全的，但是当使用一些特殊的群时证明是不安全的。Stern，Pointcheval，Malone-Lee 和 Smart[452]注意到 ECDSA 有一些在普通群模型上不具备的特性，进一步说明了在普通群模型上安全性证明的局限性。

Howgrave-Graham 和 Smart[202]首先指出敌手若知道对应于签名消息的每条消息秘密的几个比特，则能够有效地获得 DSA 或 ECDSA 的私钥（注 4.34）。Nguyen 和 Shparlinski[345，346]对它们的攻击可用于 DSA 和 ECDSA，El Mahassni，Nguyen 和 Shparlinski[130]对它们的攻击可用于 Nyberg-Rueppel 签名方案进行了形式化证明。Römer 和 Seifert[392]给出了攻击 ECDSA 的该种方法的变体。

EC-KCDSA 首先由 Lim 和 Lee [297]提出。4.4.2 节基于 ISO/IEC 15946-2 标准[212]对其进行了描述。随机预言模型由 Bellare 和 Rogaway[37]推广。Canetti，Goldreich 和 Halevi[83]提出了公钥加密和签名方案并证明了在随机预言模型下的安全性，但对于一些具体的随机函数实例是不安全的。他们的研究表明，在评估现实世界中协议的安全性时必须要格外谨慎，即便这些协议在随机预言模型下已经被证明是安全的。Pointcheval 和 Stern[378]以及 Brickell，Pointcheval，Vaudenay 和 Yung[73]证明了 DSA（以及 ECDSA）的一些变体在随机预言模型下的安全性。安全性证明并没有扩展到 DSA 和 ECDSA。在注 4.39 中提及的 KCDSA 的安全性证明由 Brickell，Pointcheval，Vaudenay 和 Yung[73]完成。

签名方案如 ECDSA 和 EC-KCDSA 有时也称为带附录的签名方案，因为在验证过程中需要输入消息 m。带（部分）消息恢复的签名方案不同于此，在验证算法中不需要输入（整个）消息。整个消息或者其中的一部分能通过签名自身恢复。在带宽资源非常紧张的环境中非常需要这样的方案。Pintsov-Vanstone（PV）签名方案[375]就是这样的签名方案的例子，能够恢复部分消息。它基于 Nyberg 和 Rueppel[350]的签名方案，Brown 和 Johnson[76]在不同假设下对其进行了广泛分析和安全性证明。Naccache 和 Stern[341]提出了另外一种提供

部分消息恢复的椭圆曲线签名方案。

4.5 节

公钥加密方案（定义 4.41）中不可辨别性（即多项式安全）的概念由 Goldwasser 和 Micali[174]构思。他们还对于语义安全性的概念进行了安全的形式化，即在计算能力范围内给定密文攻击者不能获得对应明文的任何信息，并证明了两个概念是等价的（对于选择明文攻击）。Dolev，Dwork 和 Naor[123，124]介绍了不可扩展性的概念。Rackoff 和 Simon[389]指出对于选择密文攻击需要满足这些安全性。Bellare，Desai，Pointcheval 和 Rogaway[36]研究了公钥加密方案中各种安全性概念的关系，并证明了对于选择密文攻击不可辨别性和不可扩展性是等价的。

以上的安全性定义都是针对单用户的，这里只有一个合法的实体能够解密数据，敌手的目标是危害这一过程的安全性。Bellare，Boldyreva 和 Micali[35]给出了针对多用户的公钥加密方案的安全性定义。他们研究的目的是应对诸如 Håstad 这样的攻击[195]，在该攻击中，若同样的明文 m（或者与 m 线性相关）被三个合法的实体使用基本 RSA 加密方案以加密指数 $e=3$ 进行加密，则敌手能够容易地获得明文 m。注意，只有一个合法实体的单用户公钥加密的安全性并不受 Håstad 攻击的影响。Bellare，Boldyreva 和 Micali 证明了在单用户中的安全性蕴含着多用户的安全性。

ECIES 由 Bellare 和 Rogaway[40]提出，是 ElGamal 公钥加密方案[131]的一个变体。Abdalla，Bellare 和 Rogaway[1]形式化了计算和判定 Diffie-Hellman 问题的三个变体，其难解性满足 ECIES 安全性的要求。Smart[441]将该证明改为普通群模型的情况，其中 Diffie-Hellman 问题的难解性假设被普通群的假设取代。Cramer 和 Shoup[106]在假设即使已知 ECDDHP 的有效算法 ECDHP 问题仍是难解的前提下，证明了随机预言模型下的 ECIES 的安全性。对于 Diffie-Hellman 判定问题给定一个预言器的 Diffie-Hellman 问题是 gap 问题的例子，gap 问题的概念由 Okamoto 和 Pointcheval[356]介绍。

PSEC 基于 Fujisaki 和 Okamoto[152]的研究。Cramer 和 Shoup[106]研究了密钥封装机制。Shoup 在 ISO 18033-2[215]中给出了 PSEC-KEM，DEM1 以及 PSEC 的安全性证明。

Cramer 和 Shoup[105]提出了一个基于离散对数的公钥加密方案，值得注意的是，它在标准模型下被证明是安全的（即不是在诸如普通群或随机预言模型的理想化模型下）。其安全性证明假设 Diffie-Hellman 判定问题是难解的，并对使用的杂凑函数进行了合理的假设。该方案可扩展为加密任意长度的消息，其安全性被 Shoup[426]在随机预言模型中 Diffie-Hellman 计算的假设前提下所证明，在该随机预言模型中，杂凑函数被模型化为随机函数。

Cramer-Shoup 方案的一个缺点是与其他方案相比，加密和解密过程需要更多的群求幂操作。

另外还有其他一些值得注意的基于离散对数的公钥加密方案，例如由 Pointcheval[377]及 Okamoto 和 Pointcheval[357]的通用构造方法所引申出的一些方案。这些构造方案将任何对于被动攻击不可辨别的公钥加密方案（例如基本 ElGamal 方案）转化为在随机预言模型中对于选择密文攻击的可证明不可辨别的方案。

4.6 节

Diffie 和 Hellman[121]在其划时代的论文中提出了 Diffie-Hellman 密钥协商协议。Boyd 和 Mathuria[68]提出了全面的密钥传输和密钥协商协议。也可参见 Menezes，van Oorschot 和 Vanstone[319]的第 12 章，以及 Blake-Wilson 和 Menezes[50]对 Diffie-Hellman 协议认证的概览。

最令人信服的密钥建立协议的安全性形式化定义由 Canetti 和 Krawczyk[84]给出；也可参见 Canetti 和 Krawczyk[85]。

STS 密钥协商协议（协议 4.50）由 Diffie，van Oorschot 和 Wiener[122]提出。Blake-Wilson 和 Menezes[51]针对 STS 协议，在签名和 MAC 的消息没有包含接收者的身份时，给出了一些未知的共享密钥攻击。Canetti 和 Krawczyk[84]证明了与协议 4.50 相似（但不同）的协议的安全性。

Law, Menezes, Qu, Solinas 和 Vanstone[275]研究了 ECMQV 协议（协议 4.50），给出了其安全性的启发式论点，并给出了一次传递的变体。Kaliski[237]描述了针对 ECMQV 协议两次传递变体的未知共享密钥攻击，该变体中未提供私钥确认。协议 4.51 是三次握手传递协议，可以抵抗该攻击。

许多不同的 Diffie-Hellman 密钥协商协议已被人们提出和分析。其中比较有名的是 Orman[363]的 OAKLEY 协议，Krawczyk[269]的 SKEME 协议，以及由 Harkins 和 Carrell[190]提出并由 Canetti 和 Krawczyk[85]分析的互联网密钥交换（IKE）协议。

第5章 实现问题

本章将从工程角度介绍如何在特殊环境中高效和安全地实现基于椭圆曲线的密码解决方案。我们会经常通过例子进行介绍，因为这些内容都是针对特殊平台的，且需求的竞争、物理上的约束和快速变更的硬件、笨拙的设计以及不同的目标会使这些内容变得复杂。我们的目标是从工程角度面向软件开发者和硬件设计者提供一个概览。主题和示例的选择包括通用原则或广泛使用的软件和硬件。

5.1 节将给出关于高效软件实现的选择主题。尽管涉及到特殊的平台（因此也涉及到硬件），但大多数材料都能够广泛应用。本节还包括在普通工作站上为了加速域运算而使用浮点和单指令多数据（向量）操作的注记。5.2 节将介绍有限域和椭圆曲线算术在硬件上的实现。关于安全性实现的 5.3 节将介绍各种旁门攻击。与直接从数学角度攻击安全机制不同，这类攻击试图通过伴随着物理过程或者实现结果（包括能量消耗、电子辐射、操作时间、故障分析、错误消息的分析等）而带来的信息泄漏来获取机密。特别是简单微分能量分析能够有效地针对某些设施如智能卡进行攻击，因为在这些设施中能量消耗可被精确监控。对于这些设施，为保护嵌入的秘密而添加封装是低效的（至少是昂贵的）。本节讨论一些算法上的对策，这些对策能够最小化或者减轻旁门攻击的影响，但会以牺牲一些效率作为代价。

5.1 软件实现

本节收集了涉及特殊平台的几种实现方案。在这个层面上，软件实现的决策由基于的硬件特征而来，因此本节也谈及硬件。读者不需要进行深入的理解，因为本节所给的大部分都是示例。我们集中讨论的特殊平台包括 Intel IA-32 系列（即通常我们所熟知的从 20 世纪 80 年代开始广泛使用的 *x*86 处理器）和 Sun SPARC 系列。

5.1.1 节讨论计算整数乘法的传统方法的一些缺点，特别是在 Intel Pentium 系列处理器上。5.1.2 节和 5.1.3 节给出两种类型硬件加速的技术和实现问题的概述。许多拥有硬件浮点运算能力的处理器能够用于实现素域的运算。Bernstein 提出的使用浮点的快速方法在 5.1.2 节给出。5.1.3 节考虑 Intel 和 AMD 处理器中提供的单指令多数据（SIMD）寄存器，

该寄存器能够用于加速域运算。普通 MMX 子集很适合进行二进制域运算，在 Pentium 4 上的扩展指令集，使用整数操作替代浮点方法，能用于加速素域上的乘法。5.1.4 节包括各种各样的优化技术和实现注记，包括需求、特征、缺陷以及作者使用过的开发工具中的怪事。5.1.5 节介绍域运算的时序选择。

5.1.1 整数算术

对于 p 为素数的 $\mathbb{F}_p$ 上的域运算，“传统的”实现方法是将域元素 a 表示为一系列 W 比特整数 $0 \leqslant a_i < 2^W$，其中 W 是机器的字长（如 $W = 32$），$a = \sum_{i=0}^{t-1} a_i 2^{Wi}$。教科书上的乘法使用各种扫描方法，积扫描（算法 2.10）连续地计算 $c = ab$（约减另外进行）的每个输出字。最初由 t^2 组成的乘法-累加策略使用三个寄存器的存储器 r_2, r_1, r_0，重复如下形式的迭代：

$$\begin{aligned} (uv) &\leftarrow a_i b_j \\ (\varepsilon, r_0) &\leftarrow r_0 + v \\ (\varepsilon, r_1) &\leftarrow r_1 + u + \varepsilon \\ r_2 &\leftarrow r_2 + \varepsilon \end{aligned} \tag{5.1}$$

其中 (uv) 是 a_i 和 b_j 的 $2W$ 比特乘积，ε 是进位。Karatsuba-Ofman 技术（见 2.2.2 节）减少了乘法的次数并逐渐更快，但额外的开销使得这种方法对于实践意义大小的域没有优势。

为了展示评估各种乘法策略的考虑因素，我们简要回顾 Intel Pentium 处理器系列的情况，如表 5.1 所示。Pentium 本质上是 32 位系统结构，能够并行地执行指令，因而称之为“超标量”。原始 Pentium 的流水线性能最容易描述，有两个通用整数流水线，其优化集中于原始编码保持每个流水线满负荷，服从特定流水线约束。Pentium 系列的新处理器的结构更为复杂，使用了更精致的流水线技术，如乱序操作。我们这里所讨论的内容只涉及处理器的一般特性。

Pentium 拥有的整数乘法器能够执行 32×32 位的乘法（得出 64 位的结果）。然而只有（最多）八个通用寄存器，并且乘法性能受使用的寄存器的限制。衡量性能的基本指标是指令的延迟和吞吐量，表5.2中给出了部分值。粗略地说，延迟就是一个操作得出结果前需要等待的时钟周期的数目，吞吐量是一条指令再次重复执行需要经过的时钟周期的数目[①]。注意，在这些定义下的延迟和吞吐量越小越好。

① Intel 定义延迟为执行核心完成所有由 μ 操作构成的一条 IA-32 指令所需要的时钟周期的数目，吞吐量为等待端口空闲并可以再次接受同样指令所需要的时钟周期的数目。对于大多数 IA-32 指令，一条指令的吞吐量明显少于其延迟。

表 5.1 Intel IA-32 处理器系列的发展历史和特性。所给的处理器类型还有很多变体，其特点略有不同（如老一代的 Celeron 没有缓存）。由于不同的存取速度和机制，缓存的比较变得复杂（如新 Pentium III 使用更高级的传输缓存，而一级和二级缓存大小变小）

处 理 器	年 份	MHz	缓存（KB）	主要特性
386	1985	16		首个 IA-32 系列处理器，32 位操作和并行结构
486	1989	25	L1：8	在 486 中解码和执行单元扩展为五流水线结构；处理器在每个时钟周期内能够执行一条指令
Pentium Pentium MMX	1993 1997	60 166	L1：16 L1：32	双流水线：优化的 U-V 流水线对的吞吐量在每个时钟周期内能够执行两条指令。MMX 增加了八个特殊用途的 64 位“多媒体”寄存器，支持 1，2，4，8 字节整数的向量操作
Pentium Pro Pentium II Celeron Pentium III	1995 1997 1998 1999	150 233 266 500	L1：16 L2：256，512 L1：32 L2：256，512 L2：0，128 L1：32 L2：512	P6 系统结构采用了更复杂的流水线和乱序操作。指令解码为μ操作，在一个周期内执行三个μ操作。改进了分支预测，但是预测错误所付出的代价远大于 Pentium。整数乘法延迟/吞吐量为 4/1，而 Pentium 上是 9/9。Pentium II 及新版有 MMX；III 采用了扩展的 SSE，拥有 128 位寄存器，支持单精度数值的向量操作
Pentium 4	2000	1400	L1：8 L2：256	NetBurst 系统结构的时钟速度显著提高，但很多指令需要的周期数比 P6 系列处理器更多。新的 12K μ操作“跟踪执行缓存”机制。SSE2 扩展集支持双精度浮点和 128 位整数的数据类型

表 5.2 Pentium II/III 和 Pentium 4 的指令延迟/吞吐量对比

指 令	Pentium II/III	Pentium 4
整数加，异或……	1/1	0.5/0.5
整数加，带借位减	1/1	6~8/2~3
整数乘法	4/1	14~18/3~5
浮点乘法	5/2	7/2
MMX ALU	1/1	2/2
MMX 乘法	3/1	8/2

方法(5.1)有两个性能瓶颈：指令工作对于流水线的依赖性，以及乘法操作后的延迟周期太长（特别是在 Pentium 4 上）。改进的域乘法（如同步约减）策略使用通用寄存器时会受到可用的寄存器很少和处理进位的约束，另外对于乘法结果的固定输出寄存器也受限制。可以将一些有用的存储移位指令加入方法(5.1)中以提高效率。在 Pentium II/III 上，没有重组的代码能够充分利用乘法操作后的延迟期，进行字长为 t 的乘法对于每个 32×32 乘法平均需要大约七个时钟周期。类似方法(5.1)的代码在 Pentium 4 上的效率更差。

冗余表示 通过使用不同的域表示方法，在某些情况下可以减小处理进位的开销。基本的思想是选择 $W' < W$，并将元素表示为 $a = \sum a_i 2^{W'i}$，其中 $|a_i|$ 比 $2^{W'-1}$ 稍大（这样的表示并不惟一，需要更多的字来表示一个域元素）。例如对于加法，可不需要任何进位处理。对于域乘法，选择 W' 使得 $c = ab$ 中的 $a_i b_j$ 不需要进位到第三个字。粗略地讲，其中 W' 的约 $2W'$ 位数量 $a_i b_j$ 能够存储在单（宽度）浮点寄存器中，该策略将在下节讨论。

5.1.2 浮点算术

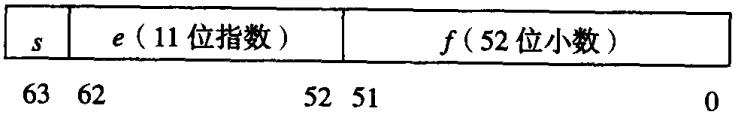
很多工作站提供的浮点硬件能够用于进行整数运算。基本的技术并不是新近提出的，尽管在普通硬件上效率的提高可能没有被完全体现。在接下来的章节中，所举的例子主要基于 Intel Pentium 系列，尽管其中大多数同样适用于其他平台。

一个形为 $2^e m$ 的有理数称为 b 位浮点数，其中 e 和 m 是满足 $|m| < 2^b$ 的整数。给定实数 z，$\mathrm{fp}_b(z)$ 表示接近于 z 的 b 位浮点数，若 $|z| \leqslant 2^{e+b}$，则 $|z - \mathrm{fp}_b(z)| \leqslant 2^{e-1}$。一个 b 位浮点数 $2^e m$ 接近于 $|z| \in ((2^{b+e} - 2^{e-1})/2, 2^{b+e} - 2^{e-1})$；下表给出了 $b = 3$ 时的例子。

e	z 区间	3 位近似值	最大误差
−1	[2−1/8, 4−1/4]	$2^{-1}m = 1a_1.a_0$	1/4
0	[4−1/4, 8−1/2]	$2^0 m = 1a_1a_0$	1/2
1	[8−1/2, 16−1]	$2^1 m = 1a_1a_00$	1

若 z 是 b 位浮点数，则 $z = \mathrm{fp}_b(z)$。受指数的限制，浮点硬件对于 b 位浮点数 x 和 y 能够找到 $\mathrm{fp}_b(x \pm y)$ 和 $\mathrm{fp}_b(xy)$，其中 b 由浮点硬件而定。

IEEE 单精度和双精度浮点格式包括一个符号位 s、指数 e 和小数 f。双精度浮点格式



得出数 $z=(-1)^s\times 2^{e-1023}\times 1.f$；有效位 $1.f$ 的标准化可将精度增加到 53 位[①]。浮点运算有时以有效位的长度描述，对于双精度来说就是 53 位。Pentium 有八个浮点寄存器，有效位的长度由控制寄存器选择。用 fp_b 来讲的话，Pentium 可有 $b\in\{24,53,64\}$（对应于 32，64 和 80 位的格式）。

浮点运算的编码要求不直接使用传统方法中的转换策略。数字以不同的格式存储，在格式之间重复转存很不经济。在整数格式中很方便的位操作（如取某些特殊位）对于浮点寄存器来说很笨拙（也很慢）。另一方面，若 $W=32$，则浮点加法比整数加法指令需要对更多的位进行操作，并且在寄存器紧张的机器如 Pentium 上会用到外部寄存器。乘法的延迟仍是个关键问题（事实上在 Pentium II/III 上比整数乘法更差，见表 5.2）；然而，这时有更多的寄存器可用，且特殊寄存器也不再被使用，使得在延迟期间能够进行有用的操作。

多精度整数乘法能够通过浮点操作和整数操作相结合来进行。若输入和输出是规范的（多字整数）格式，则在 Intel P6 系列处理器上该方法并不有效；然而 Pentium 4 的长延迟使得类似于使用 SIMD 的策略更可行（见 5.1.3 节），结合的方法已经用到了 SPARC 处理器上。

例 5.1 （SPARC 乘法）SPARC（可升级处理器系统结构）规格基于 Sun 公司设计的 RISC（精简指令集计算机）。与 Pentium 中提供一条整数乘法指令不同，32 位 SPARC-V7 处理器只有“乘法步骤”指令 MULScc，乘法实质上是重复 32 步移位-加法。

SPARC-V9 系统结构从 V8 扩展为包括 64 位地址和数据结构、加法寄存器和指令、条件和分支处理的改进以及对超标量设计的高级支持。特别是 V7 和 V8 的乘法运算，因为使用产生 64 位整数乘积的低 64 位的新 MULX 指令而被批评。在 Sun UltraSPARC 中，对通常的 32×32 乘法，MULX 相对更慢；更糟的是，该指令不能兼容能够在一个周期执行四条指令的超标量设计（受折中的限制）。

由于 MULX 的局限，GNU MP（见附录 C）的多精度库在 V9 兼容处理器上使用浮点寄存器实现了整数乘法。64 位的 a 同 b 相乘，将 a 分为 32 位的两半，b 分为 16 位的四块，对于每个 64 位的 a 执行八次浮点乘法。使用浮点加法得到两个（a 的每个 32 位字有 4 个）48 位部分乘积的和；转换成整数形式后执行剩下的操作。在 UltraSPARC I 或 II 上，计算 ab（其中每次迭代中处理的 a 和 b 都是 64 位）的主循环中的 56 条指令据称可在 14 个时钟周期内完成（每个周期 4 条指令）。

在每次域乘法中，整数和浮点之间格式的转换允许浮点变换插入存在的代码中。然而在椭圆曲线运算中，要想效率更高，则应使曲线操作中的转换次数最少。

① 对于 32 位单精度和扩展的 80 位双精度格式，会出现类似的标准化；整个 64 位有效位在扩展精度格式下保留。

P-224 上的标量乘法

我们简要介绍在 NIST 推荐的$\mathbb{F}_p$上的$p=2^{224}-2^{96}+1$的曲线中，Bernstein 提出的一种使用浮点硬件进行椭圆曲线点乘法kP的快速方法。所有性能上的改进都是在域算术（以及点倍和点加域操作的组织）中进行的。在 Pentium 上能够使用 64 位有效位，域元素表示为

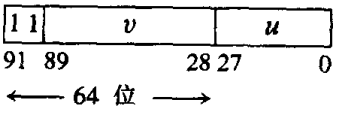
$$a=\sum_i a_i 2^{W'i}=\sum_{i=0}^{7} a_i 2^{28i},$$

其中$|a_i|$比2^{27}稍大（如 5.1.1 节的末尾所述）。与 32 位正整数的表示方法相比，该表示方法不惟一，且需要额外的一个字。域乘法需要多于64次（浮点）乘法，而传统方法只需要 49 次。但该表示方法好的方面有：可使用更多的寄存器，乘法可在任何寄存器中进行，$a_i b_j$可以在寄存器中不处理进位而直接累加。

域算术 域乘法和（部分）约减操作同步执行，从最高有效位到最低有效位按字依次计算$c=ab$。计算c_k的部分代码如下所示：

$$\begin{aligned} r_2 &\leftarrow \textstyle\sum_{i+j=k} a_i b_j \\ r_1 &\leftarrow \mathrm{fp}_{64}(r_2+\alpha_k)-\alpha_k \\ r_0 &\leftarrow r_2-r_1 \end{aligned}$$

其中r_i是浮点寄存器，$\alpha_k=3\cdot 2^{90+28k}$。简单地讲，对$\alpha_k$进行加减是从浮点数中提取二进制位的有效途径。当$k=0$时通过最近循环处理。若$r_2$是 64 位浮点数且$|r_2|<2^{90}$，则$r_1\in 2^{28}\mathbb{Z}$，$|r_0|\leqslant 2^{27}$，$r_2=r_0+r_1$。图 5.1 显示了对于$r_1$和$r_0$在$0\leqslant r_2=v\cdot 2^{28}+u<2^{90}$和$u=2^{27}$时的偶数循环处理过程。



(a) $\alpha+r_2=\alpha+v\cdot 2^{28}+u$

$u<2^{27}$ 或 $u=2^{27}$ 且 v 是偶数	$u>2^{27}$ 或 $u=2^{27}$ 且 v 是奇数
$r_1=v\cdot 2^{28}$, $r_0=u$	$r_1=(v+1)2^{28}$, $r_0=u-2^{28}$

(b) $r_1=\mathrm{fp}_{64}(r_2+\alpha)-\alpha$, $r_0=r_2-r_1$

图 5.1 $0\leqslant r_2=v\cdot 2^{28}+u<2^{90}$和$\alpha=3\cdot 2^{90}$时对于 64 位浮点数 r_2的分割。当$u=2^{27}$时，偶数循环用到了最近循环规约

对于给定的$k>7$，计算中的r_0和r_1分解成低次数的形式。第一步寻找$r_2=a_7 b_7$，然后

$c_{14}=r_0$，$c_{15}=r_1$。令 $c'_k=c_k\cdot 2^{-28k}$，考虑

$$\begin{aligned}c_k&=c'_k\cdot 2^{28k}=c'_k\cdot 2^{28(k-8)}2^{224}\\&=c'_k\cdot 2^{28(k-8)}(p+2^{96}-1)\\&\equiv c'_k\cdot 2^{28(k-5)}2^{12}-c'_k\cdot 2^{28(k-8)}\pmod p。\end{aligned}$$

这是说 $c_k\cdot 2^{-128}$ 加到 c_{k-5}，从 c_{k-8} 减去 $c_k\cdot 2^{-224}$；例如 c_{15} 分为 c_{10} 和 c_7。最后产生八个浮点数向量形式的部分约减后的乘积 $c=ab$。

曲线算术 计算 kP 的 Bernstein 点乘方法使用窗口宽度 4 的方法（无滑动），预计有 3+(15/16)(224/4)次点加[①]。在 Pentium II/III 上，点乘需要大约 730 000 个时钟周期，这比其他文献记载的实现速度明显要快。性能的提升大部分源于域乘法和平方的调度。点运算可进行组织，以使得一些操作能够通过域乘法高效地进行；例如，倍点 $(x_2,y_2,z_2)=2(x_1,y_1,z_1)$ 的域运算组织如下：

$$\delta\leftarrow z_1^2,\quad \gamma\leftarrow y_1^2\quad \beta\leftarrow x_1\gamma,\quad \alpha\leftarrow 3(x_1-\delta)(x_1+\delta)$$
$$x_2\leftarrow\alpha^2-8\beta,\quad z_2\leftarrow(y_1+z_1)^2-\gamma-\delta,\quad y_2\leftarrow\alpha(4\beta-x_2)-8\gamma^2$$

它需要三次乘法、五次平方和七次约减。将输出转化为规范形式的开销很大，但是只需要在点乘的最后进行。

编程考虑 除了一段用于设置浮点控制寄存器的代码以外，所有代码都用 C 编写。然而调度和管理寄存器是面向处理器的，类似的工作需要汇编语言完成。还需要对开发工具有一些要求。对于扩展的 80 位双精度寄存器，特别要求编译器不要将其分割为 64 位的双精度。数据排列必须适当（例如在 8 字节界限），而在一些环境中并没有适当地处理这些问题。自动变量的排列需要有额外的步骤。5.1.3 节将讨论一种使用 SIMD 整数处理能力的变更策略。

5.1.3 SIMD 和域算术

单指令多数据（SIMD）能够对向量并行执行操作。对于 Intel Pentium（见表 5.1）系列，除了老的 Pentium 和 Pentium Pro 以外都提供了这样的硬件结构。该特性最初称为多媒体应用的“MMX 技术”，由八个 64 位寄存器构成，对于 1，2，4 或 8 字节构成的向量进行操作。在随后的处理器中，该能力扩展为：Pentium III 中的 SIMD（SSE）流有 128 位寄

① 实现过程中 $k=\sum_{i=0}^{55}k_i2^{4i}$，其中 $-8\leqslant k_i<8$。在预计算阶段，对于非零的 $i\in[-8,8)$ 以 Chudnovsky 坐标 $(X:Y:Z:Z^2:Z^3)$ 存储 iP，需要三次点平方和三次点倍运算。额外的存储对于性能影响不大。

存器和单精度浮点运算单元，Pentium 4 中的 SSE2 将 SSE 扩展为双精度浮点和整数运算。AMD 开发了支持其 K6 处理器的 MMX，并在新型芯片上增加了各种扩展的版本。

本节我们将讨论在 AMD 和 Intel 处理器上使用 SIMD 能力来加速域运算。基本的思想是在 32 位机器上用特殊用途寄存器来实现快速 64 位操作。对于二进制域，普通 MMX 命令集能够用于乘法和求逆的加速。对于素域，SSE2 扩展指令集（Pentium 4 特有）提供了取代 5.1.2 节中的浮点方法的另外一种途径。

使用 MMX 的二进制域运算

Pentium 和 AMD 处理器上的八个 64 位 MMX 寄存器能够相对容易地用于加速二进制域 $\mathbb{F}_{2^m}$ 上的操作。尽管支持的函数有限，还是提供了二进制域上需要的基本移位和异或操作。本节研究 MMX 命令集对于域乘法和求逆的性能和缺点。

与仅使用通用 32 位寄存器的代码相比，64 位寄存器能够将性能提高两倍。实践中的结果取决于算法和编码方法。实现过程可能混用传统编码和 MMX 编码，这样只有算法的一部分能够得益于 64 位寄存器。比较操作产生一组掩码向量而不是状态寄存器，并且不直接支持基于数据的分支。MMX 寄存器不能用于存储器寻址。另一方面，Pentium 只有八个通用寄存器，因此有效地使用外部寄存器能够间接提高通用寄存器的管理效率。如表 5.2 中所注，在 Pentium II/III 上使用 MMX 时没有延迟和吞吐量的损耗；在 Pentium 4 上需要更细致地考虑调度问题。

域乘法 使用 MMX 能够有效实现带约减的复合乘法（算法 2.36）。考虑约减多项式为 $f(z)=z^{163}+z^7+z^6+z^3+1$ 的域 $\mathbb{F}_{2^{163}}$。预计算的步骤 1 使用 MMX，累加器 C 包含六个 MMX 寄存器；处理输入 a 的过程由通用寄存器完成。算法能够很好地改编为使用 64 位寄存器，因为只需要进行简单的异或和移位操作。对于 MMX 寄存器和整个都在寄存器中的累加器 C，这里没有比较。与传统方法相比，乘法运算速度大约是其两倍。

域求逆 算法 2.48（Euclidean 算法的变体）实现了求逆。与乘法相比，求逆算法中的一些操作在 MMX 上实现效率较低。在步骤 3.1 中需要计算次数，步骤 3.3 因为非常数值的移位而需要载入外部寄存器。我们测试了两种方案。第一种只在 g_1 和 g_2 使用 MMX，使用传统编码求解 u 和 v 的长度和次数。第二种方案通过对所有的四个变量都使用 MMX 而获得了少许的性能提高。u 和 v 的长度通过 32 位计数器求取，以便执行次数计算时效率更高（通过求根后的一半的 32 位，再使用传统方法计算次数）。与不带 MMX 的方法相比，性能提高了 1.5 倍。

编程考虑 与 5.1.2 节描述的使用浮点寄存器的需要不同，使用 MMX 可以有效地独立

于一些特殊程序如域乘法——椭圆曲线方案中的其他代码不需要变动。在 C 中的实现可以通过引用汇编语言段或者内部指令来完成。汇编语言代码能够对寄存器分配和调度进行控制，用来实现算法 2.36。内部指令编程有些地方类似于汇编语言，但由编译器管理寄存器的分配和执行优化。求逆的程序由内部指令编码。

SIMD 和素域算术

Pentium III 有八个 128 位 SIMD 寄存器，Pentium 4 上的 SSE2 扩展集支持双精度浮点数和 64 位整数的向量操作。与 5.1.2 节描述的浮点实现相比，使用整数 SSE2 能够有效地独立于一些特殊程序如域乘法。

SSE2 硬件上的乘法在传统指令上（32 位输入，64 位结果）不会增加操作数的最大宽度；然而有更多的寄存器能够参与乘法运算，乘法的延迟更小，且乘积能够使用 64 位寄存器来求和。使用传统编码处理进位是一个瓶颈，但是因为运算操作设置能够便利地使用的条件代码而直接支持。SSE2 寄存器没有专门面向这种类型的编码设计，在外部测试进位的开销很大。算法 2.9 中乘法扫描操作数的实现直接通过标量 SSE2 操作，因为不需要在加法中考虑进位。该方法对于在内部乘积操作 $(UV) \leftarrow C[i+j] + A[i] \cdot B[j] + U$ 中的每个乘法有两次加法和一次移位。通过修改算法 2.10 中的很多乘法开销的乘积扫描策略，能够减少加法和移位的总数。为了避免测试进位，输入中的一个或者两个表示为 $a = \sum a_i 2^{W'i}$ 的形式，其中 $W' < 32$，使得乘积能够在 64 位寄存器中累加。

例 5.2 （使用 SSE2 整数操作的乘法）假设输入由七个 32 位字（如 5.1.2 节讨论的 P-224）所表示的整数组成。算法 2.9 的标量实现要在 SSE2 寄存器中执行 49 次乘法、84 次加法和 49 次移位。若输入分段为 28 位，则算法 2.10 要执行 64 次乘法、63 次加法和 15 次移位，来得到 16 个 28 位段的乘积。

GNU MP 的多精度库（见附录 C）使用 11 条指令内部循环的操作数扫描方法。代码非常紧凑，因为可处理不同长度的输入而很通用。若使用适于乘法时间的参数，则执行时间可以与使用更复杂代码获得的时间相比。然而在现实测试中，乘积扫描方法使用针对于 7 字的代码时要快 20%，尽管输入必须分割为 28 位段，输出必须重排为 32 位字。对于 7 字输入和 14 字输出（32 位字），直接用 SSE2 整数实现需要约 325 个时钟周期，这比传统方法的时间的一半还少（在表 5.2 中，由于指令延迟，传统方法在 Pentium 4 上更慢）。

5.1.4 实现平台

本节选择一些关于优化技术和平台特征的注记，其中一些是作者使用过的开发环境。

使用高级语言时，通常选择C程序语言的编译器细节注记。即使打算手工开发汇编的实现，高级语言中的原型也可以加快算法的开发和比较。因此对于估计汇编语言的性能提供有用的性能和其他信息的比较是必要的。

通常优化的考虑

我们给出能广泛应用的基本性能考虑和技术。

循环展开 在提升性能的通常策略中，将循环展开是最基本和最有效的一种。将循环扩展使得每轮都能进行更多的操作，减少了指令执行的数目，但是增加了代码长度。优化的规模越大代码就越长，极端的情况是将循环完全展开。例如算法2.36的复合乘法能够通过w位窗口在一个外循环中有效实现，完全展开的内部循环只执行加法和移位。

典型地，可以通过用户选择影响编译器执行循环展开的程度。对目前的编译器技术现状，自动方法不能取代面向程序员的方法，特别是展开会带来降低数据相关性的代码改变。

数据寻址 在寄存器紧缺的机器如Intel Pentium上，为了对数据寻址而对寄存器的开销对于优化不利。将数据复制到堆栈，可允许同其他变量使用同样的寄存器来进行寻址。注意，使用普通基本寄存器在转移增加时会使指令变长（在指令长度不定的处理器如Pentium上）。

重复代码 对一些算法，复制代码或写特殊情况段是很有效的。例如Euclidean算法的各种版本调用对于存储域元素的数组内容的反复交换。这时可通过对数组复制内容或者交换指针来管理；但使用除变量名字外本质上都相同的独立代码段能够获得更快的性能。

类似地，特殊情况代码段能够有效减少条件和其他操作的数目。例如Euclidean算法的变体，数组在执行过程中会增长或收缩。若能够有效探知长度，就能编写不同的代码段，并在长度于特定范围内时执行合适的代码段。算法2.50中的殆逆算法的情况更特殊，其中两个变量增长，两个缩短。若使用t个字来表示域元素，则需要t^2个针对不同长度的代码段。在Intel Pentium和Sun SPARC上的测试表明，该算法确实需要这样的改进才能与算法2.48相竞争。

使用“跳伞”策略能够特别有效地复制代码。其基本思想是去掉处理不相似或新数据的代码，并在遇到这些数据时转移到不同的程序执行。该方法在最坏情况下性能很差，但可以带来显著的优化（部分原因是需要更少的代码）。在Euclidean算法2.48中，步骤3.3有大量移位后的“不相似”数据，使用该技术非常有效。

复制代码或特殊情况代码段会使编码大小显著增长。平台特征和应用约束可能会对这样的策略有所限制。

分支预测失误 条件表达式会明显降低优化和性能，特别是预测结果错误时。例如 Intel Pentium 系列在 P6 处理器中改进的分支预测，预测失误时的代价很大。在测试包含很多条件表达式的程序的时间时，特别要注意考虑各种情况。若用法不同，则重复调用同样数据的时间可能使用常用方法实现 OEF 上的运算，以及 3.6.2 节中描述的在二进制域上求解 $x^2+x=c$ 的程序，这方面的体现特别明显，这是因为对于实际的数据分支预测可能非常差。

为了减少错误分支预测的频率而采用的技术包括算法的改变、查表、专用指令。对于 2.4.2 节中的 OEF，执行很多子域操作的通常方法被有较少条件表达式的算法取代。查表是广泛使用的一种方法，若表的大小容易处理，则效果会很好（查表能够减少代码，所以将编码和查表相结合能够比没有查表的方法节约存储）。对于求解 $x^2+x=c$ 的算法 3.86，这种方法很有效，能够将步骤 3 中的条件消除，并能同时对多位进行处理。最后，Pentium II 及后来的处理器提供了特殊的指令，包括条件转移和其他指令，以便在相关的指令间消除分支。

汇编代码 性能考虑、捷径、寄存器分配和平台特性在汇编语言中通常能正确处理编码的临界段。若要支持各种平台，则对整个程序编码时需要做很多工作——即使对于同一处理器系列，为了得到最好的性能也需要不同的方案。

考虑方法(5.1)中的乘法-累加段，用汇编语言编码有两个原因：一些编译器对于输入 W 位得出 $2W$ 位的乘积不能高效处理，需要使用处理进位标志的指令而不是对进位的外部测试。在更长的段中，对于寄存器分配也可能比编译器做得更好。

一些编译器支持的嵌入汇编语言，很适于插入短的程序段。例如 Euclidean 算法 2.48 需要多项式次数的计算。相对更快的方法是一旦非零字被确定，就使用二元查找和查表。一些处理器有快速的“位扫描”指令集：Pentium 有单指令（bsr 和 bsf）来寻找一个字的最高和最低有效位的位置[①]。类似地，Sun 在 SPARC 上使用汉明重量指令从右进行快速位扫描。GNU C 和 Intel 编译器能够很好地嵌入这样的代码，因为可以直接同周围的代码交互操作。与之相比，微软的编译器对这样的交互操作只提供有限的支持。

编译器特性和缺陷

本节剩余的注记是关于平台细节的。编译器涉及到 GNU C (gcc-2.95)，Intel (6.0)，Microsoft (6.0)和 Sun Workshop (6U2)，对于 Intel Pentium 系列产生 32 位代码，对于 Sun UltraSPARC 产生 64 位代码。

① 使用 Pentium 系列的位扫描指令的时钟周期数各不相同。浮点硬件能够用于提供位扫描的替代策略。

标量与数组 一些编译器在使用数组而不是标量时，会产生更慢的代码（尽管在编译时已知数组索引）。在编译器测试中，GNU C 显示了该优化的弱点。

指令调度 与 Intel 和 Sun 编译器相比，GNU C 在 Pentium 和 SPARC 平台上的指令调度能力较弱，但是能够通过对源代码相对较小的改动产生一些更好的序列。特别是在 SPARC 上较小地重组代码，对于算法 2.36 的测试的运行时间就明显不同。Sun Workshop 编译器对这样的改动就不敏感，通常产生的代码也更快。

在 Intel 处理器上，因为可用的寄存器很少，所以使用通用寄存器的调度和其他优化就会受限制。常见的策略是允许段指针（ebp）作为通用寄存器来使用；在 GNU C 中称为“fomit 段指针”。

队列 处理器需要对数据进行排列（例如 32 位整数排成 4 字节），访问没有排列的数据可能会出错或更慢。这一点对于双精度浮点数和 SIMD 操作的数据格外重要，因为一些环境不能正确地管理需要的队列。这些缺点可以在随后的开发工具中纠正。无论如何，自动（堆栈）变量的排列可能需要额外的步骤。

缺陷 尽管编译器的测试已成熟，但找到弱点还是相对容易。例如对算法 2.36 的 64 位实现中的一小段代码的改变就触发了 Sun Workshop 编译器上优化缺陷的出现，导致将乘以 16 改为移动 4 位，因而在该平台上操作会更慢。工作区包括汇编输出的预处理或使用更弱的优化设置。

在微软编译器中嵌入 C 代码会出现更明显的优化问题；特别是一个短 OEF 程序中的乘法有时会被函数调用取代。该缺陷会导致代码更大且速度更慢。广泛使用的微软编译器产生的代码同 Intel 编译器产生的代码相比更具竞争力（假设未遇到缺陷）。然而，嵌入汇编语言与周围的代码交互操作的能力的局限与 GNU C 或者 Intel 编译器相比是设计上的一个弱点，后者另外的优点是能够在 Unix 系统中使用。

5.1.5 时间

我们选择给出在工作站中通常使用的 Intel Pentium 系列处理器和 Sun UltraSPARC 的一些域操作时间。时间主要考虑 NIST 推荐的二进制域和素域（见 A.2 节），另外会给出 OEF（见 2.4 节）上的一些数据，以便于对比。

注意时间可能有误，且时间受程序员的才能和努力、编译器的选择、获取数据的具体方法等因素的影响很大。这里给出的数据都是按同一误差尺度采集的。无论如何，时间是算法分析中最根本的，因为粗略的操作计数并不足以获取平台的特性。这里给出的详细时间与其他实现的时间相对独立。

表 5.3 至表 5.5 给出了 NIST 推荐的二进制域和素域同选择的一个 OEF 的基本比较。两个平台的二进制域上的求逆和乘法的时间在表 5.6 中给出，比较了编译器、求逆算法以及 32 位和 64 位代码。Intel Pentium III 上的 64 位代码使用了特殊用途寄存器。该能力在 Pentium 4 上被扩展，表 5.7 包括了使用浮点寄存器的方法通过这些寄存器执行素域乘法的时间。

表 5.3 在 800 MHz Intel Pentium III 上执行域运算的时间（μs）。二进制域$\mathbb{F}_{2^{163}} = \mathbb{F}_2[z]/(z^{163}+z^7+z^6+z^3+1)$和$p_{192}=2^{192}-2^{64}-1$的素域$\mathbb{F}_{p_{192}}$是 NIST 推荐的（见 A.2 节）。最右一列是素数$p=2^{31}-1$的最佳扩域$\mathbb{F}_{p^6}=\mathbb{F}_p[z]/(z^6-7)$

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{p_{192}}$[a]	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{(2^{31}-1)^6}$
加法	0.04	0.18	0.07	0.06
约减				
快速约减	0.11[b]	0.25[c]	0.11[c]	N/A
Barrett 约减（算法 2.14）	N/A	1.55[d]	0.49	N/A
乘法（包括快速约减）	1.30[e]	0.57[d,f]	0.42[f]	0.40[g]
平方（包括快速约减）	0.20[h]	—	0.36[i]	0.32[g]
求逆	10.5[j]	58.3[k]	25.2[k]	2.9[l]
I/M	8.1	102.3	60.0	7.3

[a] 主要用 C 编写。[b] 算法 2.41。[c] 算法 2.27。[d] 使用 32×32 乘法和加法。[e] 算法 2.36。[f] 算法 2.10。[g] 例 2.56。[h] 算法 2.39。[i] 算法 2.13。[j] 算法 2.48。[k] 算法 2.22。[l] 算法 2.59。

域运算的比较

表 5.3 给出了 NIST 推荐的最小二进制域和素域同 OEF 的基本比较。它们是约减多项式 $f(z)=z^{163}+z^7+z^6+z^3+1$的二进制域 $\mathbb{F}_{2^{163}}$，$p_{192}=2^{192}-2^{64}-1$的素域$\mathbb{F}_{p_{192}}$，素数 $p=2^{31}-1$和约减多项式 $f(z)=z^6-7$的 OEF$\mathbb{F}_{p^6}$。通过使用一系列伪随机生成的域元素得到实际的分支预测错误代价，时间包括诸如函数调用等框架工作。使用 Intel 编译器版本 6 和 Netwide 汇编语言（NASM），在 Intel Pentium III 上的 Linux 2.2 操作系统下运行。

除了在求逆中多项式次数计算使用一行汇编语言段之外，所有二进制域算法的代码都用 C 编写。为了使用硬件乘法器对 32 位输入产生 64 位乘积，也为了直接处理进位，在素域和 OEF 上需要汇编代码，在传统方法中这两个目的对于性能都是至关重要的。表 5.3 中$\mathbb{F}_{p_{192}}$的第一列给出了主要由 C 编写的代码的执行时间。大多数情况下与汇编相比它有明显的时间开销。乘法程序以嵌入的汇编语言段来使用三字累加器进行32×32乘法。若排除约减，则时间很接近于使用汇编语言得到的时间，这表明 Intel 编译器插入短嵌入汇编语言段的处理能力很好。

约减 Barrett 约减未使用 NIST 素数的特殊形式，结果可以视为随机 192 位素数的约减的估计开销。与特殊的素数相比，该估计表明约减在域乘法的运行时间中占很大一部分，从而驱使我们使用 Montgomery（见 2.2.4 节）和其他乘法方法。由于处理进位的笨拙方法中的许多条件转移，C 版本的快速约减算法的性能降低很明显。

表 5.4 在 800 MHz Intel Pentium III 上执行二进制域运算的时间（μs），包括约减为规范形式的时间。域是 NIST 推荐的（见 A.2 节），约减多项式依次是 $z^{163}+z^7+z^6+z^3+1$，$z^{233}+z^{74}+1$，$z^{283}+z^{12}+z^7+z^5+1$，$z^{409}+z^{87}+1$ 和 $z^{571}+z^{10}+z^5+z^2+1$

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
加法	0.04	0.04	0.04	0.06	0.07
约减（算法 2.41~2.45）	0.11	0.13	0.19	0.14	0.33
乘法（算法 2.36）	1.30	2.27	2.92	5.53	10.23
平方（算法 2.39）	0.20	0.23	0.32	0.31	0.56
求逆（算法 2.48）	10.5	18.6	28.2	53.9	96.4
I/M	8.1	8.2	9.7	9.8	9.4

表 5.5 在 800 MHz Intel Pentium III 上执行素域运算的时间（μs），包括约减为规范形式的时间。域是 NIST 推荐的（见 A.2 节），素数依次是 $p_{192}=2^{192}-2^{64}-1$，$p_{224}=2^{224}-2^{96}+1$，$p_{256}=2^{256}-2^{224}+2^{192}+2^{96}-1$，$p_{384}=2^{384}-2^{128}-2^{96}+2^{32}-1$ 和 $p_{521}=2^{521}-1$

	$\mathbb{F}_{p_{192}}$	$\mathbb{F}_{p_{224}}$	$\mathbb{F}_{p_{256}}$	$\mathbb{F}_{p_{384}}$	$\mathbb{F}_{p_{521}}$
加法	0.07	0.07	0.08	0.10	0.10
约减（算法 2.27~2.31）	0.11	0.12	0.30	0.38	0.20
乘法（算法 2.10）	0.42	0.52	0.81	1.47	2.32
平方（算法 2.13）	0.36	0.44	0.71	1.23	1.87
求逆（算法 2.22）	25.2	34.3	44.3	96.3	163.8
I/M	60.0	70.0	54.7	65.5	70.6

OEF 表5.3中的最右一列OEF $\mathbb{F}_{(2^{31}-1)^6}$ 同 $\mathbb{F}_{p_{192}}$ 的大小大致相同。乘法通过使用与 $\mathbb{F}_{p_{192}}$ 使用的方法类似的累加方法完成（例 2.56），运行时间有可比性。在 OEF 上求逆明显更快，这与预计相吻合。

NIST 域 表 5.4 和表 5.5 给出了 NIST 推荐的二进制域和素域上的运行时间。注意，大域的优化受限于 $\mathbb{F}_{2^{163}}$ 和 $\mathbb{F}_{p_{192}}$ 所用的技术。对于较小的域，Karatsuba-Ofman 方法很差，而在较大的域中我们并未进行详细测试。

二进制域上的乘法和求逆

椭圆曲线上的点乘（见 3.3 节）、域求逆的开销相对于域乘法特别重要。本节给出 NIST 二进制域在两个平台上的大概比较（差距相对很小）。比较了 2.3.6 节讨论的三种求逆方法，给出了 32 位和 64 位代码的执行时间。对于不同的编译器，结果差别较大。

表 5.6 给出了两个常用平台 Inte Pentium III 和 Sun UltraSPARC IIe 的时间比较。这两种处理器都能进行 32 位和 64 位操作，尽管只有 UltraSPARC 是真正的 64 位处理器。Pentium III 中的 64 位操作是通过 Pentium MMX（见表 5.1）中的单指令多数据（SIMD）寄存器来实现的。求逆算法是算法 2.48 中扩展的 Euclidean 算法（EEA）、算法 2.49 中的二元 Euclidean 算法（BEA）以及算法 2.50 中的殆逆算法（AIA）。例子使用的域是从 NIST 推荐的域中选取的，约减多项式为 $f(z)=z^{163}+z^7+z^6+z^3+1$，$f(z)=z^{233}+z^{74}+1$。这两个多项式都能进行快速约减，但只有后者适合殆逆算法。基于 comb 方法（算法 2.36）的域乘法在这些平台中速度最快。使用的宽度为 4，时间包括了约减。除了 EEA 中有 MMX 代码和一行汇编语言段外，其余的算法都全部用 C 编码。

表 5.6 Inte Pentium III 和 Sun UltraSPARC IIe 的乘法和求逆时间。编译器是 GNU C 2.95（gcc），Intel 6（icc），Sun Workshop 6U2（cc）。在“mmx”中使用了 64 位“多媒体”寄存器。求逆和乘法的比率（*I/M*）采用最优的求逆时间

	Pentium III（800 MHz）			SPARC（500 MHz）		
	32 位		64 位	32 位		64 位
算法	gcc	icc	mmx	gcc	cc	cc
$\mathbb{F}_{2^{163}}$ 上的运算						
乘法	1.8	1.3	0.7	1.9	1.8	0.9
Euclidean 算法	10.9	10.5	7.1	21.4	14.8	–
二元 Euclidean 算法	20.7	16.0	–	16.8	14.9	10.6
殆逆	16.4	15.2	–	22.6	15.2	–
I/M	6.1	8.1	9.8	8.8	8.2	12.1
$\mathbb{F}_{2^{233}}$ 上的运算						
乘法	3.0	2.3	–	4.0	2.9	1.7
Euclidean 算法	18.3	18.8	–	45.5	25.7	–
二元 Euclidean 算法	36.2	28.9	–	42.0	34.0	16.9
殆逆	22.7	20.1	–	36.8	24.7	–
I/M	6.1	8.2	–	9.2	8.5	9.9

表中的一些结果与预计相符，例如$\mathbb{F}_{2^{233}}$上的殆逆时间。另一些显示了不同平台和不同编译器之间的差异。表 5.6 仍存在一些表面上的矛盾，但我们相信采用最快时间能对这些平台上的求逆和乘法的开销进行估计。

除法 使用修改的 BEA（见 2.3.6 节）进行除法的情况并不很多。对于 32 位代码，除非 EEA 或 AIA 变得能够有效地进行除法，在 SPARC 上对于 $\mathbb{F}_{2^{163}}$ 只支持使用 BEA 类除法。此外在多数情况下比率 I/M 最少为 8，因此使用除法算法的节省会少于 10%。鉴于这样的比率，椭圆曲线的方法会选择减少求逆的次数，因此点乘 kP 的节省会明显少于 10%。

另一方面，如果在使用基于倍-加的点乘方法中仅使用仿射运算，那么即使 I/M 明显大于 5，快速除法仍然很适用。如果选择的算法是 BEA，那么除法和求逆的开销本质上相同。

实现注记 5.1.4 节中包含了这里用到的实现的通常编程考虑。特别地，在 Sun SPARC 上使用 gcc 是为了获得可接受的乘法时间，以使代码更加“gcc 友好”。在求逆代码中也针对 gcc 进行了有限调整。优化求逆代码是很麻烦的，部分原因是在这个层面粗略计数操作数常常不能获得处理器或编译器的全面特性。

多媒体寄存器 Intel Pentium 系列（除老 Pentium 和 Pentium Pro 以外）和 AMD 处理器拥有八个 64 位“多媒体”寄存器，在标记“mmx”的列中使用了这些寄存器。5.1.3 节中讨论了在域运算中如何利用这些处理能力的方法。

EEA 算法 2.48 需要多项式的次数计算。在 SPARC 上，一旦确定了所需的非零字，就可通过二元搜索和查表寻找次数。在 Pentium 上通过嵌入的汇编语言，用位扫描指令（bsr）寻找一个字中的最高有效位的位置，对于求逆时间有大约 15%的提高。

BEA 算法 2.49 通过 t 段分割有效地处理长度 u 和 v。与步骤 3.3 中的次数计算不同，使用了更简单的相关字的比较。

AIA 算法 2.50 能够有效处理长度 g_1 和 g_2（除长度 u 和 v 之外）。总共使用了相似的 t^2 段代码，除非 t 很小，否则代码的规模将扩展得很大。同 BEA 一样，使用了类似的比较来取代次数计算。注意，只有 $\mathbb{F}_{2^{233}}$ 的约减多项式才适于殆逆算法。

素域乘法方法

对于素域，传统的域乘法方法常因为硬件整数乘法器和进位传播的局限而不可行。UltraSPARC 和 Pentium 系列处理器都有这样的局限。Intel Pentium 4 事实上比先前的 Pentium 处理器在一些操作中更慢（以处理器时钟来比较）。例如表 5.5（取自 Pentium III）以及表

5.7（取自 Pentium 4）中在 $\mathbb{F}_{p224}$ 上使用算法 2.10 面向 Pentium II/III 编码的域乘法。除去 Pentium 4 的两倍时钟速度优势以后，实际的时间比 Pentium III 上获得的时间更慢。

Karatsuba-Ofman 基于 Karatsuba-Ofman 的方法在 PentiumII/III 上对于这种大小的域与传统方法相比并没有优势。表 5.7 中包含了 Pentium 4 上使用例 2.12 给出的深度 2 的方法。通过针对 Pentium 4 的代码的额外调整，传统方法和 Karatsuba-Ofman 方法都能得到改进；但两种方法都劣于下面讨论的使用特殊寄存器的方法。

表 5.7 在 Pentium 4 1.7GHz 上执行 224 位 NIST 素数 $p_{224}=2^{224}-2^{96}+1$ 的域 $\mathbb{F}_{p224}$ 上的乘法。浮点方法的时间包括（部分）约减到八个浮点数的时间，但不包括与规范形式之间的转换；其他方法的时间都包括约减

$\mathbb{F}_{p224}$ 上的乘法	时间（μs）
传统整数（算法 2.10）	0.62
Karatsuba-Ofman（例 2.12）	0.82
SIMD（例 5.2）	0.27
浮点（5.1.2 节中的 P-224）	0.20[a]

[a] 不包括与规范形式之间转换。

浮点运算 在工作站上通常都有浮点硬件，在下面的策略中可以使用。基本思想是利用快速浮点处理能力来使用合适的域元素表示进行整数运算，这在 5.1.2 节中已详细讨论。在诸如椭圆曲线点乘的应用中，若将曲线操作写成新的域表示形式，则在整数和浮点数形式之间的转换的高额开销对于整个计算会有一定的影响。5.1.2 节中对于 NIST 推荐的 $p224=2^{224}-2^{96}+1$ 的素域 $\mathbb{F}_{p224}$ 已经提到了这种策略。在 Pentium 4 上使用浮点方法计算乘法的时间在表 5.7 中给出。注意时间包括（部分）约减到八个浮点数（每个长度大概 28 位）的时间，但是不包括转换为规范的约减形式的高额开销。

SIMD 在 Pentium 4 上还可以使用单指令多数据（SIMD）寄存器来进行快速乘法。二进制域运算的普通 MMX 命令集在前面的章节已经介绍，Pentium 4 上的 SSE2 扩展集适合于 64 位整数向量的整数操作。5.1.3 节更详细地讨论了特殊寄存器。与浮点方法相比，使用 SIMD 寄存器的域元素表示和规范形式之间的转换相对开销较小，并且将 SIMD 代码插入到大框架中的难度也较低。表 5.7 中的 SIMD 方法的时间包括转换和约减到规范形式的时间。

5.2 硬件实现

在一些应用中，为达到必需的安全级别而用软件实现的椭圆曲线加密方案无法满足预期的性能要求。在这种情况下，设计和构造硬件加速器有利于满足性能的要求。本节将介绍椭圆曲线系统的硬件实现，主要的设计问题将在 5.2.1 节讨论，5.2.2 节将介绍有限域处理器的体系结构。下面是硬件设计中的一些基本概念的概述。

门（Gate）：门是一个用于修改其输入并产生单个输出的小型电路。最常见的门有两个输入（但也有多于两个输入的）。门是现代计算设备的基本组成部分。大多数通用门是非门（NOT，输入的反），与非门（NAND，两个输入的逻辑与结果的求反），或非门（NOR，两个输入的逻辑或结果的求反），与门（AND）以及或门（OR）。门数典型地是指与两个输入的与非门等效的门的数量。

VLSI（Very Large Scale Integration）：是指由超过 10 000 个门组成的电路。在 VHDL、Verilog 或其他一些硬件描述语言中，一条 VLSI 电路从描述开始，这些描述要么被编入产生电路的信息中（通称为综合），要么被编入运行在通用机上的源代码中（通称为仿真）。VLSI 电路的设计需要在由信号传播引起的电路延迟和功率损耗之间权衡，物理电路布局也会影响这两个因素。其他可得工具包括用于部件摆放的布局编辑器和用于调整设计的定时分析工具。这些定制的设计可能会花费大量的时间、金钱和其他资源。

FPGA（Field-Programmable Gate Array）：可编程门阵列由很多相互连接的逻辑部件组成，每个部件通常包括多个门。这些部件通过合理应用关闭或开启特定的电子路径而转换为电路。本质上，FPGA是可编程的，它允许在其生成后电路仍能被创建或修改，且这些变化是可逆的（因为其“可编程域”的特点）。一个 FPGA 可以大到由 20 000 个或更多的门组成。最初 FPGA 被介绍成为一种原型方法，后来被越来越多地用于创建性能经常优于运行在通用处理器上的二进制代码的特定应用电路。在具体厂商的工具中使用的程序设计类似于创建 VLSI 电路的程序设计。

门阵列：（Gate Array）：门队列由一组规则的彼此相互连接的逻辑部件组成，每个逻辑部件通常不只包含一个门。通过部件之间合理的融合连接构成了电路，这个过程是不可逆的。随着具有相当灵活性的 FPGA 的出现，门队列技术很少被使用。

ASIC（Application-specific integrated circuit）：专用集成电路是关于 VLSI 或门阵列方面的术语。

多路复用器（Multiplexor）：多路复用器是一个有着多个输入、单个输出的设备，它有一个控制由哪个输入产生输出的控制器。多路复用器提供了对电路的条件控制。

流水线技术（Pipelining）：流水线技术允许在当前计算过程完成之前开始第二个计算过程。

并行处理（Parallel Processing）：并行处理是一种允许两个或多个计算过程同时进行的技术。

5.2.1 设计标准

点乘操作决定了椭圆曲线加密协议的执行时间。点乘的有效实现分为三个不同的层次：

1. 有限域算术（见第 2 章）。
2. 椭圆曲线点加和倍点（见 3.2 节）。
3. 点乘方法（见 3.3 节）。

点乘操作具有层次性，点乘方法处于较高层，而基础有限域算术则位于底层。如图 5.2 所示，这种层次关系也引入到了协议中。例如，若以硬件完全实现 ECDSA 签名（见 4.4.1 节），则设备上的惟一输入就是待签名的信息，而惟一的输出就是信息的签名。

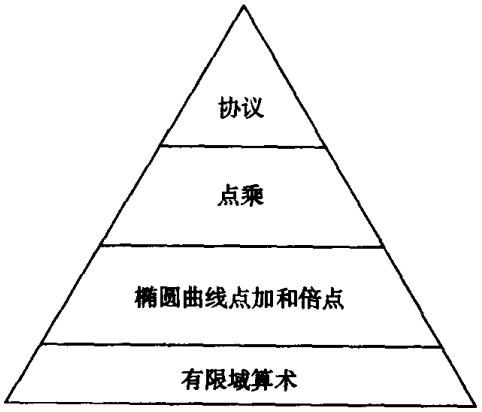


图 5.2 椭圆曲线密码方案中的操作层次

硬件设计的关键点是确定在硅片中实现的层次结构，显然，有限域算术也应在硬件设计中实现，有可能的话，仅为有限域算术设计一个硬件加速器，并使用现有的微处理器来执行椭圆曲线点算术的高级功能。需要注意的是，有效的有限域乘法器未必能产生有效的点乘，在这个层次结构中，所有层都需要优化。

在硬件中实现点加、倍点和点乘为 ECC 处理器提供了更高的效率，但这也增加了其复

杂性。在所有的情况中，高效的算法和硬件结构的结合都是必需的。

图 5.3 显示了一种高性能的处理器。除了程序和数据内存外，还有三个主要组件：算术逻辑单元（AU）、算术单元控制器（AUC）和主控制器（MC）。算术逻辑单元执行加法、平方、乘法和求逆等基本运算，它由算术单元控制器控制。算术控制单元执行椭圆曲线的点加和倍点操作。主控制器与主机系统相互结合，协同执行点乘方法。

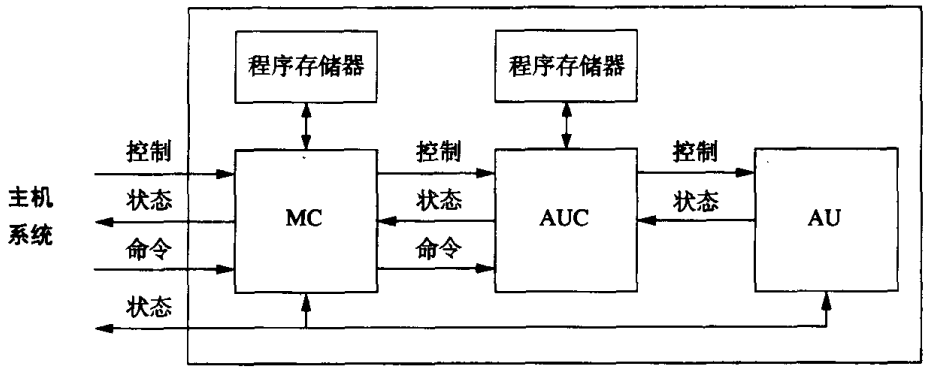


图 5.3 椭圆曲线处理器的体系结构

现在需要考虑的是当随机选择一个整数 k 时，高性能处理器是如何处理 kP 计算的。由主机提供整数 k 和 P 的（仿射）坐标，主机要求处理器生成 kP，其中整数 k 被载入至 MC 中，而 P 的坐标被载入至 AU 中。MC 命令 AUC 进行初始化操作，包括将 P 的仿射坐标转换为点加和倍点公式所需的投影坐标。MC 会扫描 k 的比特位，并命令 AUC 执行适当的椭圆曲线操作，AUC 将顺次指使 AU 执行适当的有限域操作。当 k 的所有比特位被处理后，MC 将指使 AUC 把返回的结果转换为仿射坐标，主机则从 AU 的寄存器中读取 kP 值。拥有两个控制器有两方面的重要性，即并行处理的能力和流水线操作。MC也会使用数据存储的能力来实现更有效地利用预计算来计算 kP 值的算法（见 3.3 节）。

选择硬件设计的标准

下面是一些在硬件设计过程中不得不考虑的问题。好的设计强调对目标平台、操作环境、性能和安全要求的全面理解。

1. *成本*总是硬件设计者考虑的一个重要问题，它由下面所有的标准来驱动。
2. *硬件和软件*。是否必须考虑在软件实现之上再选择硬件加速器呢?
3. *吞吐量*。安装在服务器上的装置有可能要求在每秒钟执行上百或上千次椭圆曲线操作，尽管为掌上电脑设计的这种装置只要求执行很小一部分这样的操作。
4. *复杂性*。装置实现的层次越多，其电路将越复杂，这使得它在定制的 VLSI 装置上

有更多的硅体或者成为一个更大的 FPGA，其结果便是更高的成本。

5. *适应性*。相关问题包括装置在二进制域和素域的曲线上执行计算的能力。
6. *算法的灵活性*。很多加密协议要求在每一会话基础上（如 SSL）协商加密算法。可重构的硬件实现有个诱人的特点，即它几乎不影响系统的性能。
7. *功耗*。依赖于装置的操作环境，功耗可能是也可能不是一个主要的问题。例如，尽管服务器能够承担起更多的功耗，但非接触式智能卡在加密操作方面的功耗仍受到限制。
8. *安全性*在任何设计中都是需要重要考虑的一部分。若装置仅被设计用来执行点加和倍点，则在点乘 kP 过程中它将被与随机数 k 相关联的比特位激活。若没有精心设计的整体架构，则 k 的比特位可能会因旁门攻击而泄漏。在装置的操作环境中，应该考虑到基于时限、能量分析和电磁辐射等的攻击（见 5.3 节）。
9. *整体系统结构*。若在整个系统中，微处理器有足够的空闲周期来处理上述有限域算术（见图 5.2）的协议功能，则根据其他标准，有理由设计出仅供有限域算术使用的装置。
10. *实现平台*。在定制的 VLSI、门阵列或 FPGA 等平台上均能实现。相对于 VLSI 和门阵列装置，FPGA 的每个单元成本比较高，而 VLSI 实现的设计成本也是比较高的。
11. *可伸缩性*。若期望装置能够提供各种不同级别的安全（如在 A.2 节中实现的所有的 NIST 曲线），则必须设计出底层的有限域处理器，以适应不同的域大小。

这些设计标准的相对重要性更多地依赖于应用。例如，对高端服务器上的硬件成本的考虑就比低端设备考虑得少。表 5.8 显示了在高端和低端设备上实现的设计标准的优先级。

表 5.8 硬件设计标准的优先级

高端设备		低端设备	
高优先级	低优先级	高优先级	低优先级
吞吐量	成本	成本	吞吐量
安全性	功耗	硬件与软件	适应性
可伸缩性	复杂性	复杂性	算法灵活性
系统结构		功耗	可伸缩性
实现平台		安全性	
算法灵活性		系统结构	
适应性		实现平台	
硬件与软件			

5.2.2 域算术处理器

本节将讲述在二进制域$\mathbb{F}_{2^m}$上执行加法、乘法、平方和求逆操作的硬件电路。相对于素域$\mathbb{F}_p$，在$\mathbb{F}_{2^m}$上使用硬件实现这些操作更为容易，因为在$\mathbb{F}_{2^m}$上，按位加不需要任何进位传送。另外，不同于$\mathbb{F}_{2^m}$的情况，在$\mathbb{F}_p$上实现平方操作所花的成本和普通的乘法所花的成本基本是一样的。而在$\mathbb{F}_p$上实现平方的成本高于在$\mathbb{F}_{2^m}$上的成本，因而在$\mathbb{F}_p$上使用乘法的求逆（参看下面在$\mathbb{F}_{2^m}$上实现的描述）的速度也相对较低。

加法

回顾一下 2.3.1 节，在二进制域$\mathbb{F}_{2^m}$上的元素的加法是按位执行的，且没有进位传送，因此与素域$\mathbb{F}_p$相比，在$\mathbb{F}_{2^m}$上用硬件实现加法是相当简单的。

乘法

我们讨论一下在二进制域$\mathbb{F}_{2^m}$上乘法组件的硬件电路的设计。我们仅考虑$\mathbb{F}_{2^m}$上的元素由多项式表示的情况。若$f(z)$是约减多项式，则我们可写成

$$f(z)=z^m+r(z)，\text{其中}\deg r\leqslant m-1。$$

而且，若$r(z)=r_{m-1}z^{m-1}+\cdots+r_2z^2+r_1z+r_0$，则我们可用二进制向量

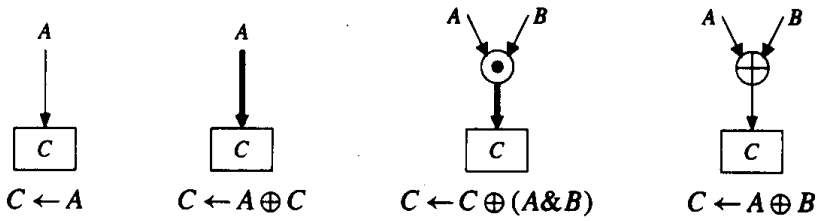
$$r=(r_{m-1},\ldots,r_2,r_1,r_0)$$

表示$r(z)$。

若乘法器在每个时钟周期产生出乘积的一个位，则称它为位串行的。若乘法器在每个时钟周期产生出乘积的多个位，则称它为数字串行的。我们将描述下面三种情况下的位串行乘法器：

(i) 任意约减多项式的固定大小的域。
(ii) 固定约减多项式的固定大小的域。
(iii) （任意或固定约减多项式的）可变大小的域。

我们也将描述下面第四种情况下的数字串行乘法器：
(iv) 固定约减多项式的固定大小的域。
如图 5.4 至图 5.11 所示，用下面的符号表示对位 A，B，C 的操作：



(i) **任意约减多项式的固定大小的域** 描述被乘数$a \in \mathbb{F}_{2^m}$与乘数$b \in \mathbb{F}_{2^m}$的乘法的 算法 5.3 在处理 b 时是从左（高位）到右（低位）处理的，这种乘法器称为高位优先乘法器（MSB）。图 5.4 描述了当$m = 5$时的这种乘法器，b 是一个移位寄存器，c 也是一个移位寄存器，且其低位补 0。高位优先乘法器能在$\mathbb{F}_{2^m}$上的 m 个时钟周期内执行一次乘法运算。

算法 5.3 $\mathbb{F}_{2^m}$的高位优先（MSB）乘法器

输入：$a = (a_{m-1}, \ldots, a_1, a_0)$，$b = (b_{m-1}, \ldots, b_1, b_0) \in \mathbb{F}_{2^m}$和约减多项式$f(z) = z^m + r(z)$。

输出：$c = a \cdot b$。

1. 令$c \leftarrow 0$。
2. 对于 i 从$m-1$降序到 0，执行
 2.1 $c \leftarrow$左移$(c) + c_{m-1} r$。
 2.2 $c \leftarrow c + b_i a$。
3. 返回(c)。

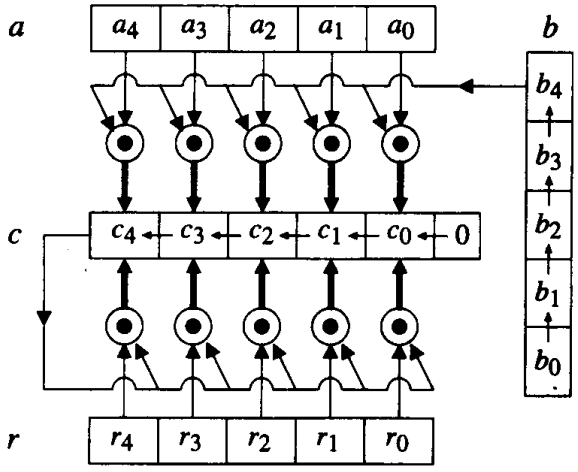


图 5.4 $\mathbb{F}_{2^5}$上的高位优先（MSB）乘法器

描述被乘数$a \in \mathbb{F}_{2^m}$与乘数$b \in \mathbb{F}_{2^m}$的乘法的算法 5.4 在处理 b 时是从右（低位）到左（高位）处理的，图 5.5 中描述的这种乘法器称为低位优先乘法器（LSB）。

算法 5.4 $\mathbb{F}_{2^m}$的低位优先（LSB）乘法器

输入：$a=(a_{m-1},\ldots,a_1,a_0)$，$b=(b_{m-1},\ldots,b_1,b_0)\in\mathbb{F}_{2^m}$和约减多项式$f(z)=z^m+r(z)$。

输出：$c=a\cdot b$。

1. 令$c\leftarrow 0$。
2. 对于i从0升序到$m-1$，执行
 2.1 $c\leftarrow c+b_i a$。
 2.2 $a\leftarrow$左移$(a)+a_{m-1}r$。
3. 返回(c)。

MSB 和 LSB 乘法器之间的一个不同是在乘法过程中，图 5.4 中的四个寄存器中的两个上的内容在改变，而图 5.5 中的四个寄存器中的三个上的内容在改变。换言之，每个时钟周期内 MSB 乘法器仅有两个寄存器，而 LSB 乘法器上有三个寄存器。

(ii) **固定约减多项式的固定大小的域** 若约减多项式$f(z)$是固定的，并选定为三项式或五项式，则乘法器的设计将大大地降低其复杂度，因为不再需要保存约减多项式的寄存器。图 5.6 举例说明了$\mathbb{F}_{2^5}$上的固定约减多项式$f(z)=z^5+z^2+1$的高位优先乘法器。

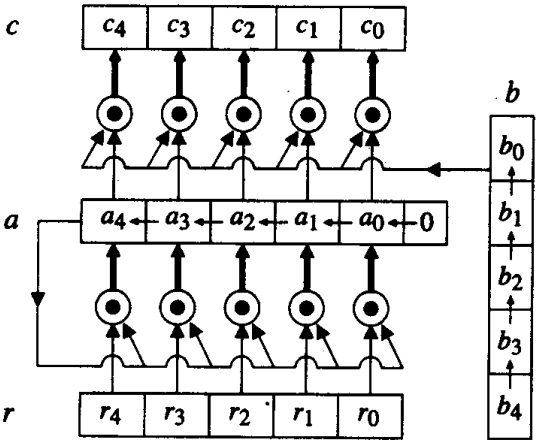


图 5.5 $\mathbb{F}_{2^5}$的低位优先（LSB）乘法器

(iii) **可变大小的域** 图 5.4 所示的高位优先乘法器可扩展为域$\mathbb{F}_{2^m}$上的元素的乘法，其中$m\in\{m_1,m_2,\ldots,m_t\}$，$m_1<m_2<\cdots<m_t$，每个寄存器的长度为$m_t$。图 5.7 描述了在任意域$\mathbb{F}_{2^m}$上$m\in\{1,2,\ldots,10\}$和任意约减多项式中实现乘法的高位优先乘法器，注意在每个时钟周期内只有寄存器b和c的内容在发生变化。控制器按从高到低的顺序载入a，b和r的比特位，并将无用位置0。虽然无用位也被时钟所驱动，但由于无用位的内容未发生变化，

所以其功耗几乎为零。

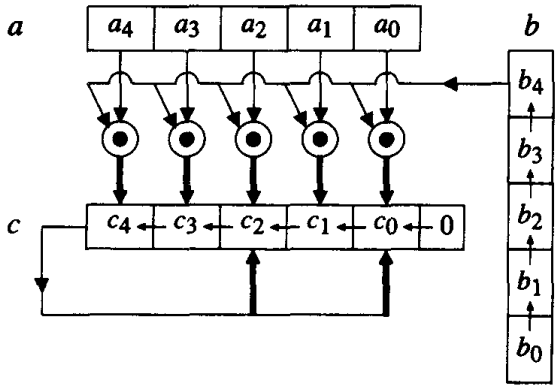


图 5.6 固定约减多项式 $f(z)=z^5+z^2+1$的高位优先（MSB）乘法器

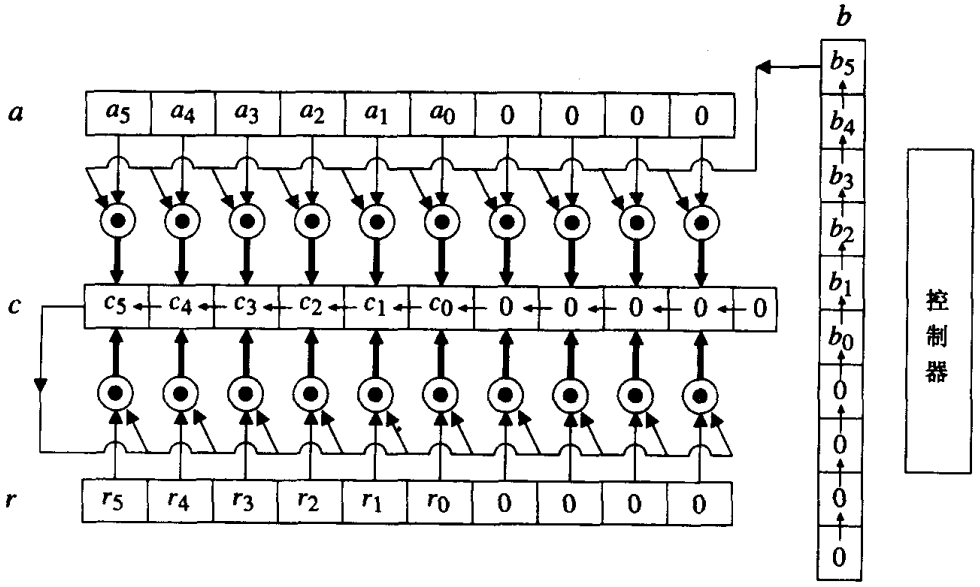


图 5.7 域$\mathbb{F}_{2^m}$上$1\leqslant m\leqslant 10$的高位优先乘法器，图示为$\mathbb{F}_{2^6}$的乘法器

若每个域有固定的约减多项式，最好是三项式或五项式，则电路将被简化。图5.8描述了$\mathbb{F}_{2^5}$，$\mathbb{F}_{2^7}$和$\mathbb{F}_{2^{10}}$的可变大小域的高位优先乘法器，它们的固定约减多项式分别为z^5+z^2+1，z^7+z+1和$z^{10}+z^3+1$。一个多路复用器用于选定预期的域，控制器用于载入寄存器和控制多路复用器。

(iv) **固定的约减多项式的固定大小的域的数字串行乘法器** 我们现在考虑$\mathbb{F}_{2^m}$上a和b的二元乘法，其中b表示为含有$l=\lceil m/k\rceil$位的多项式

$$b=\sum_{i=0}^{l-1}B_i z^{ki},$$

其中每个B_i都是次数至多为$k-1$的二元多项式。乘积$a\cdot b$可表示为

$$\begin{aligned}a\cdot b&=a\left(\sum_{i=0}^{l-1}B_i z^{ki}\right)\bmod f(z)\\&=\left(\sum_{i=0}^{l-1}B_i(az^{ki}\bmod f(z))\right)\bmod f(z),\end{aligned}$$

其中$f(z)$是$\mathbb{F}_{2^m}$上的约减多项式。由此得出的算法5.5是一个数字串行乘法器。

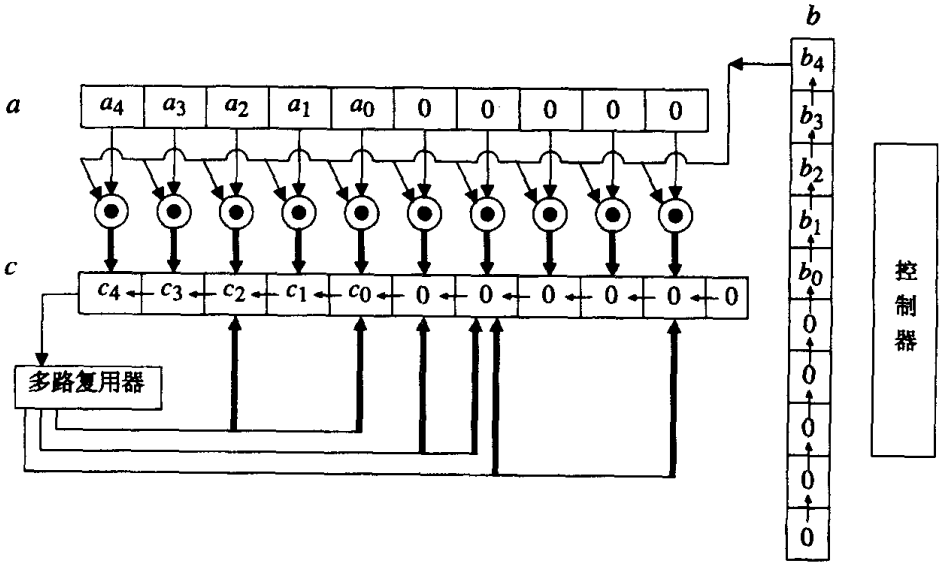


图5.8 域$\mathbb{F}_{2^5}$，$\mathbb{F}_{2^7}$和$\mathbb{F}_{2^{10}}$上约减多项式为z^5+z^2+1，z^7+z+1和$z^{10}+z^3+1$的高位优先乘法器，图示为$\mathbb{F}_{2^5}$的乘法器

算法5.5 $\mathbb{F}_{2^m}$上的数字串行乘法器

输入：$a=\sum_{i=0}^{m-1}a_i z^i\in\mathbb{F}_{2^m}$，$b=\sum_{i=0}^{l-1}B_i z^{ki}\in\mathbb{F}_{2^m}$，约减多项式$f(z)$。

输出：$c=a\cdot b$。

1. 令$c\leftarrow 0$。
2. 对于i从0升序到$l-1$，执行
 2.1 $c\leftarrow c+B_i a$。
 2.2 $a\leftarrow a\cdot z^k \bmod f(z)$。
3. 返回$(c \bmod f(z))$。

执行算法 5.5 的硬件电路包括保存被乘数 a 的移位寄存器、保存乘数 b 的移位寄存器和保存 c 的累加寄存器（不是移位寄存器）。保存 a 和 b 的寄存器的长度均为 m 位，而保存 c 的寄存器的长度为$(m+k-1)$位。在第i次循环时，a 为 $az^{ki} \bmod f(z)$，乘积 $B_i \cdot (az^{ki} \bmod f(z))$称为数位乘法，数位乘法的结果至多为$m+k-1$位长，且结果在累加器 c 中被执行异或运算。若电路在单个时钟内完成对 $az^{ki} \bmod f(z)$和 $B_i \cdot (az^{ki} \bmod f(z))$的计算，则整个乘法运算将在 l 个时钟周期内完成。当电路复杂度增加 k 时，乘法运算将以 k 倍的速度加速完成。

图 5.9 显示了约减多项式为$f(z)=z^5+z^2+1$的$\mathbb{F}_{2^5}$上的 2 位数乘法器的一个寄存器，在这个例子中，$k=2$，$l=3$。除图 5.9 所示电路的互连之外，图 5.10 给出了数位乘法的电路和 c 寄存器中最终约减多项式$f(z)$的电路的互连。最终简化的互连需要电路复用器。

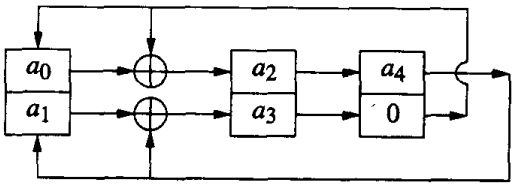


图 5.9 计算$az^{ki} \bmod f(z)$的电路，其中$f(z)=z^5+z^2+1$，$k=2$

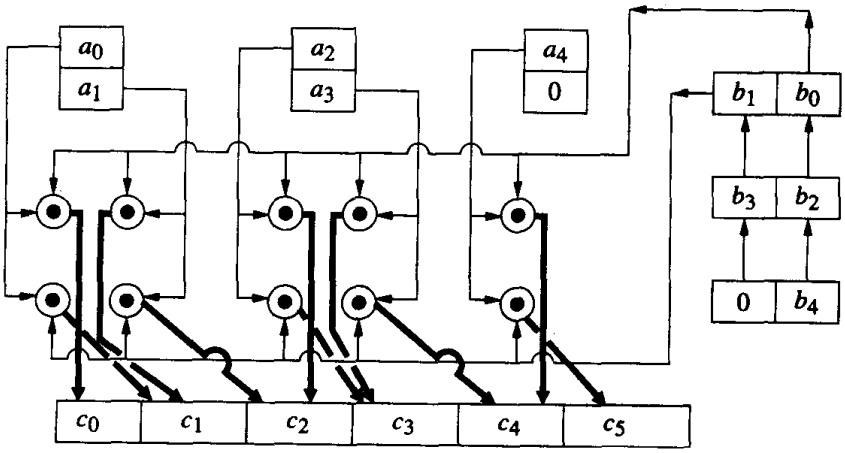


图 5.10 $f(z)=z^5+z^2+1$的$\mathbb{F}_{2^5}$上的 2 位数乘法器

平方

平方当然能够使用上面描述的任何乘法器来执行。若约减多项式$f(z)$是固定的三项式或五项式，则可以设计一个电路在单独一个时钟周期（对于位串行乘法器来说是 m 个时钟周期）内执行一次平方操作。此外，平方电路将只对乘法电路增加很少的复杂度。当使用

乘法（见下文）进行求逆时，只需要一个时钟周期的平方电路就显得很重要。

例如，考虑约减多项式 $f(z)=z^7+z+1$ 的域 $\mathbb{F}_{2^7}$。若 $a=a_6z^6+a_5z^5+a_4z^4+a_3z^3+a_2z^2+a_1z+a_0$，则

$$\begin{aligned}c&=a^2\\&=a_6z^{12}+a_5z^{10}+a_4z^8+a_3z^6+a_2z^4+a_1z^2+a_0\\&=(a_6+a_3)z^6+a_6z^5+(a_5+a_2)z^4+a_5z^3+(a_4+a_1)z^2+a_4z+a_0。\end{aligned}$$

图 5.11 描述了一个平方电路。

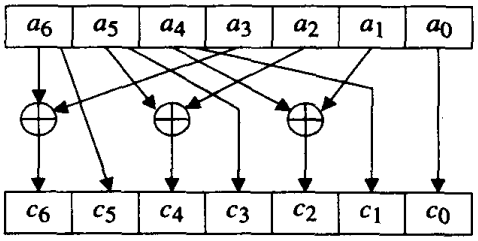


图 5.11 固定的约减多项式 $f(z)=z^7+z+1$ 的域 $\mathbb{F}_{2^7}$ 上的平方电路

求逆

在硬件实现中最困难的有限域操作是求逆。求逆算法有两个基本类型：基于扩展的 Euclidean 算法及其变体（见 2.3.6 节）和使用域乘法的算法。使用乘法的求逆并没有明显增加硬件设计的复杂度，但若需要频繁执行时，则对性能的影响会很大。这就是大多数硬件（和同样情况的软件）设计者选择仿射投影坐标的原因。在控制器中必须加入额外的功能，但是不需要对核心电路进行全面的修改。如果使用仿射坐标，那么求逆无疑是性能的瓶颈，因此基于扩展的 Euclidean 算法的求逆电路就成为必要的条件。这样的电路对于核心电路和控制器都将增加更多的复杂度。由于增加的复杂度使得扩展的 Euclidean 算法的求逆实现似乎并不可取，因此我们的注意力主要放在使用乘法的求逆方法上。

令 a 是 $\mathbb{F}_{2^m}$ 上的非零元素。使用乘法的求逆利用

$$a^{-1}=a^{2^m-2}。\tag{5.2}$$

因为 $2^m-2=\sum_{i=1}^{m-1}2^i$，所以我们有

$$a^{-1}=a^{\sum_{i=1}^{m-1}2^i}=\prod_{i=1}^{m-1}a^{2^i}。\tag{5.3}$$

因此 a^{-1} 能够通过 $m-1$ 次平方和 $m-2$ 次乘法计算出来。我们接下来展示如何减少乘法的次

数。首先观察

$$a^{-1}=a^{2^m-2}=(a^{2^{m-1}-1})^2。$$

因此，一旦求出$a^{2^{m-1}-1}$的值，a^{-1}就能够用一次平方计算出来。现在若m是奇数，则

$$2^{m-1}-1=(2^{(m-1)/2}-1)(2^{(m-1)/2}+1)。 \tag{5.4}$$

若我们令

$$b=a^{2^{(m-1)/2}-1},$$

则由式(5.4)我们有

$$a^{2^{m-1}-1}=b\cdot b^{2^{(m-1)/2}}。$$

因此，一旦求出b的值，$a^{2^{m-1}-1}$就能够用一次乘法和$(m-1)/2$次平方计算出来。类似地，若m是偶数，则

$$2^{m-1}-1=2(2^{m-2}-1)+1=2(2^{(m-2)/2}-1)(2^{(m-2)/2}+1)+1。 \tag{5.5}$$

若我们令

$$c=a^{2^{(m-2)/2}-1},$$

则由式(5.5)我们有

$$a^{2^{m-1}-1}=a\cdot\left(c\cdot c^{2^{(m-2)/2}}\right)^2。$$

因此，一旦求出c的值，$a^{2^{m-1}-1}$就能够用两次乘法和$m/2$次平方计算出来。重复递归该过程最终计算出a^{-1}。这一过程中全部的乘法运算次数为

$$\lfloor \mathrm{lb}(m-1)\rfloor+w(m-1)-1, \tag{5.6}$$

其中$w(m-1)$表示$m-1$的二元表示中 1 的个数，而整个平方运算次数是$m-1$。算法 5.6 给出了当m是奇数时的求逆过程。

算法 5.6 $\mathbb{F}_{2^m}$上求逆（m为奇数）

输入：非零元素$a\in\mathbb{F}_{2^m}$。

输出：a^{-1}。

1. $A\leftarrow a^2, B\leftarrow 1, x\leftarrow(m-1)/2$。
2. 当$x\neq 0$时，执行

2.1 $A \leftarrow A \cdot A^{2^x}$。

2.2 若 x 为偶数，则 $x \leftarrow x/2$；

否则，$B \leftarrow B \cdot A$, $A \leftarrow A^2$, $x \leftarrow (x-1)/2$。

3. 返回(B)。

表 5.9 给出了使用算法 5.6 在 NIST 二进制域 $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{409}}$ 和 $\mathbb{F}_{2^{571}}$ 上求逆计算所需要的平方和乘法的次数。步骤 2.2 中 A 的最后一个平方是不需要的，因此没有包括在操作次数统计中。

表 5.9 使用算法 5.6 在二进制域 $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{409}}$ 和 $\mathbb{F}_{2^{571}}$ 上求逆所需要的操作次数统计

m	$\lfloor \mathrm{lb}(m-1) \rfloor$	$w(m-1)$	乘法	平方
163	7	3	9	162
233	7	4	10	232
283	8	4	11	282
409	8	4	11	408
571	9	5	13	570

5.3 实现的安全

在评估一个密码协议的安全性时，通常假设敌手知道协议的全部细节，掌握所有的公钥，只是缺少私钥的信息。此外敌手能够截取合法通信方之间传递的一些数据，甚至可以对这些数据进行控制（例如对于一个签名方案通过选择消息进行选择明文攻击，或者对于一个公钥加密方案通过选择密文进行选择密文攻击）。敌手试图通过求解被假定困难的问题，或者寻找协议设计中的一些缺陷，来危及协议的安全。

在这种传统安全模型中考虑的攻击都是从协议规范的数学基础来进行分析的。近年来，研究者越来越注意到可能利用实现和操作环境的特性的攻击。这样的旁门攻击利用协议执行时的信息泄漏，这在传统的安全模型中没有考虑到。例如当使用智能卡执行诸如解密和签名的生成等私钥操作时，敌手可能监视能量消耗或者电磁辐射。敌手还能够测量执行一个密码操作的时间，或者分析当遇到特定错误时密码设备的举动。在实践中很容易收集旁门信息，因此在全面评估系统的安全性时确定旁门攻击的威胁是至关重要的。

需要注意的是，在一些环境中特定的旁门攻击并不构成现实的威胁。例如，当密码设施是从外部不可信的来源处获取能量的智能卡时，对于密码设施测量能量消耗的攻击似乎

是可行的。然而，若密码设施是位于一个安全的办公室中的工作站，则能量攻击就没什么威胁了。

本节的目标是提供旁门攻击和相应对策的介绍。我们在 5.3.1 节、5.3.2 节、5.3.3 节、5.3.4 节和 5.3.5 节中分别讨论能量分析攻击、电磁分析攻击、错误消息分析、故障分析攻击和时间分析攻击。提出的相应对策包括算法的、基于软件的、基于硬件的或以上方法的组合。没有哪个对策能够保证防御所有的旁门攻击。此外，这些对策可能会降低密码的计算速度，需要消耗更多的存储或硬件资源。高效和安全地在诸如智能卡等设备上实现密码协议是正在进行的充满挑战的研究问题，需要密码学者和工程师两方面的关注。

5.3.1 能量分析攻击

CMOS（互补金属氧化物半导体）逻辑是用于微处理器、存储器和专用集成电路（ASIC）的核心半导体技术。CMOS 逻辑的基本组成单元是反相器或者非门，如图 5.12 所示。反相器由两个晶体管组成，一个为 P 型，一个为 N 型，作为控制电压的开关。高电压信号表示逻辑“1”，低电压信号表示逻辑“0”。若输入电压 Vin 低，则 P 型晶体管导通（即开关闭合），而 N 型晶体管不导通；这时从电压到输出有通路，于是 Vout 高。相反地，若输入电压 Vin 高，则 P 型晶体管不导通，而 N 型晶体管导通；这时从输出到地有通路，于是 Vout 低。当反相器转换状态时，在一个很短的期间内两个晶体管同时导通。这使得从电源到地有短电路。当连接在反相器输出的内部电容充电或者放电时，也会有电流。

在一个时钟周期内电流只有很少一部分通过 CMOS 设备的门——这些门在执行特殊指令时激活。因此当设备执行一系列复杂指令时，可预测设备的能量消耗连续地改变。

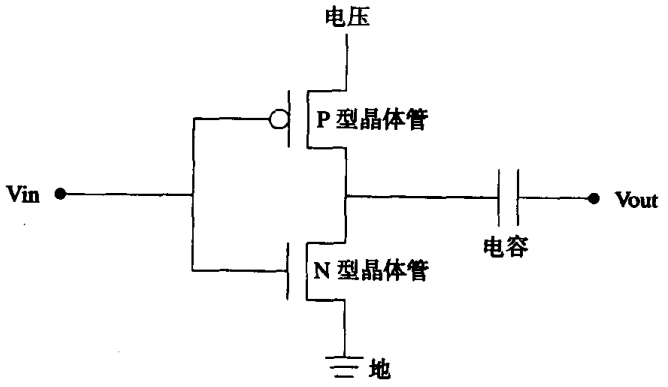


图 5.12 CMOS 逻辑反相器

如果对于设备供应的能量是恒定电压，那么设备消耗的能量同电流成比例。对于电流和消

耗的能量的测量，能够通过在电源处设置一个串联电阻，用示波器来测量通过电阻的电压的差别来确定。这样就可以绘出能量轨迹，显示出在每个时钟周期设备的能量消耗。

能量分析攻击的假设前提是能量轨迹与设备执行的指令相关，并与处理的操作数的值相关。这样检查能量轨迹能够暴露正在执行的指令和寄存器中数据内容的信息。当设备正在执行私钥加密操作时，敌手就有可能推导出私钥。

简单能量分析

在简单能量分析（SPA）攻击中，敌手通过直接检查单独的私钥操作中的能量轨迹就能推导出关于私钥的信息材料。椭圆曲线点乘算法的实现易受这样的攻击，因为点加和倍点的公式差别很大，直接通过能量轨迹就能将两者区分开。图 5.13 显示了在素域上的椭圆曲线的一系列加法（S）和倍点（D）操作的能量轨迹。点的表示使用雅可比坐标（见 3.2.1 节），这时一个加法操作的时间明显比一个倍点的时间要长。

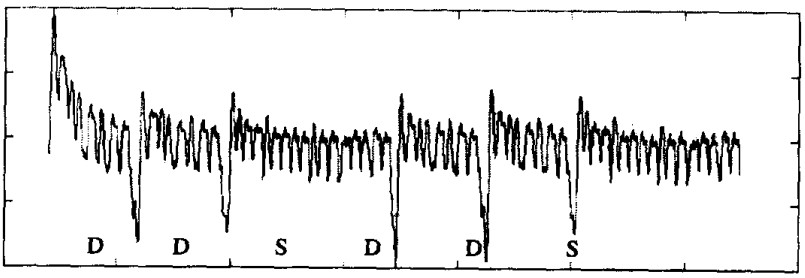


图 5.13 素域上椭圆曲线的一系列加法（S）和倍点（D）操作的能量轨迹。点的表示使用雅可比坐标。轨迹从 SC140 DSP 处理器内核获取

例如，考虑一个设备在 ECDSA 签名生成（算法 4.29）过程中执行点乘 kP。这里 P 是公开的椭圆曲线上的点，k 是秘密的整数。从单独的每条消息秘密 k 和相对应的消息与签名能够容易地求出长期私钥（注 4.34）。首先假设对于点乘使用二元方法中的一个（算法 3.26 和算法 3.27）。如果对于点乘的能量轨迹的检测暴露出倍点和加法操作的序列，那么就能马上知道 k 的各个独立的位。假设现在使用更复杂的点乘方法，考虑二元 NAF 方法（算法 3.31）。如果能量轨迹暴露出倍点和加法操作的序列，那么敌手就能够知道 NAF(k)为 0 的位，从而得出 k 的相关信息。

关于算法如何使用和实现的知识能够使 SPA 攻击更便利。任何由密钥位决定执行路线的实现都潜在地易受该攻击。

对策 目前已提出了有很多种抵抗 SPA 攻击的技术。这些对策包括算法的修改、软件的实现、硬件的实现或者以上方法的结合。对策的有效性很大程度上依赖于硬件平台和操

作环境的特征以及敌手的能力，因而必须基于各种情况具体评估。例如，图 5.14 显示了在素域上椭圆曲线的一系列加法（S）和倍点（D）操作的能量轨迹。在算法的加法和倍点过程中插入了虚假的操作，插入的方式是序列中倍点操作的基本操作的重复次数刚好是加法操作的两倍。与图 5.13 相比，通过检查图 5.14 的能量轨迹来辨别加法和倍点操作会变得不可能。

没有哪个提出的对策能够保证提供充足的保护。另外，能够抵抗 SPA 攻击并不能保证抵抗其他旁门攻击，如差分能量分析和电磁分析攻击。对于 SPA 攻击，目前不可能提供最好的通用对策。我们这里只给出一个例子，其他方法将在本章的注释一节中给出。

算法 5.7 是从左到右二元点乘方法的修改，它提供了针对 SPA 攻击的增强抵抗。在主循环中加入了虚假操作，使得每轮循环执行同样的基本椭圆曲线操作（一次倍点和一次加法）。因此从能量轨迹推导出的倍点和加法的序列不会暴露任何关于 k 的各个位的信息。与大多数对策算法一样，安全性的增强是以速度变慢为代价的。

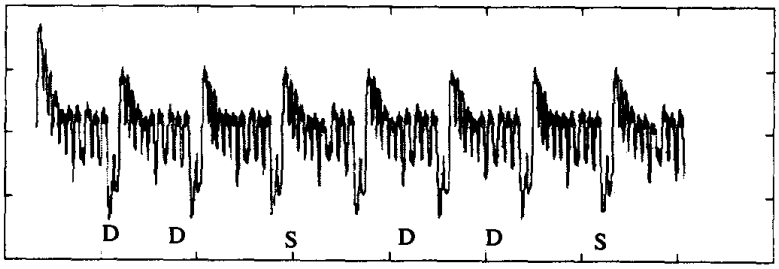


图 5.14 素域上椭圆曲线的一系列加法（S）和倍点（D）操作的能量轨迹。点的表示使用雅可比坐标。通过在加法和倍点算法中插入虚假操作（与算法 5.13 相比），达成对于 SPA 的抵抗。轨迹从 SC140 DSP 处理器内核获取

算法 5.7 抵抗 SPA 的从左到右二元点乘

输入：$k=(k_{t-1},\ldots,k_1,k_0)_2$，$P\in E(\mathbb{F}_q)$。

输出：kP。

1. $Q_0\leftarrow\infty$。
2. 对于 i 从 t–1 到 0，执行
 2.1 $Q_0\leftarrow 2Q_0$。
 2.2 $Q_1\leftarrow Q_0+P$。
 2.3 $Q_0\leftarrow Q_{k_i}$。
3. 返回(Q_0)。

差分能量分析

差分能量分析（DPA）攻击利用能量消耗的变化与操作的数据的值之间的关系进行。与不同指令序列相关的变化相比，这样的变化很小，因而可能被噪声和测量误差所混淆。采集能量轨迹时使用了统计的方法，这样可以减少噪声和放大信号的差别。

为了执行 DPA 攻击，敌手首先选择在密码操作执行时遇到的内部变量 V，V 包含有输入消息 m，未知密钥的一部分 k' 决定 V 的值。测定函数 $V = f(k', m)$ 称为选择函数。为简便起见，假设 V 是单个比特。敌手从执行密码操作的设备上收集一定数目的能量轨迹（例如几千个），然后对 k' 进行猜想，并将能量轨迹按照预测的比特 V 的值分成两部分。对每部分的能量轨迹取平均数，平均的差别就被划分出来，称为差分轨迹。基本思想是 V 的值会对能量轨迹有影响（可能很小）。因此，若对 k' 的猜想不正确，则能量轨迹的分块本质上总是随机的，可以预计差分轨迹变得扁平。另一方面，若对 k' 的猜想正确，则两个能量轨迹的平均数会有显著差别；V 影响差分轨迹的划分。重复这一过程（使用同一采集的能量轨迹）直到 k' 被确定。

这样的思想随后将用于 DPA 攻击算法 5.7 的抗 SPA 的点乘方法。该攻击表明 SPA 对策不一定能抵抗 DPA 攻击。

在椭圆曲线签名方案如 ECDSA（算法 4.29）中，DPA 攻击对于点乘并不实用，因为当基点 P 固定时，对每个签名私钥 k 是不同的。但是该攻击能够用于椭圆曲线加密和密钥协商协议中的点乘。例如对于 ECIES 解密过程（算法 4.43）中的点乘，乘数是 $k = hd$，其中 d 是长期私钥，h 是余因子，基点是 $P = R$，而 R 是包含在密文中的点。

假设现在敌手收集了密码设备使用算法 5.7 计算 kP_1，$kP_2, \ldots, kP_r$ 的能量轨迹。敌手知道 $P_1, P_2, \ldots, P_r$，想要确定 k。若 $Q_0 = \infty$，则步骤 2.1 中的倍点操作很小，于是通过检查单独的能量轨迹与一般的倍点操作相区别。于是敌手容易判断 k 的最左位是 1。假设 $k_{t-1} = 1$。在步骤 2 的第一个循环做如下操作（$i = t-1$）：$Q_0 \leftarrow \infty$，$Q_1 \leftarrow P$，$Q_0 \leftarrow P$。在步骤 2 的第二个循环做如下操作（$i = t-2$）：$Q_0 \leftarrow 2P$，$Q_1 \leftarrow 3P$，$Q_0 \leftarrow 2P$（若 $k_{t-2} = 0$）或 $Q_0 \leftarrow 3P$（若 $k_{t-2} = 1$）。接下来的循环中当且仅当 $k_{t-2} = 0$ 时点 $4P$ 被算出。选择点的二元表示的一位，将能量轨迹按照选择的 $4P_i$ 的位是 0 还是 1 分为两部分。按一般 DPA 攻击的描述，密钥部分是 $k' = k_{t-2}$，$m = P_i$，选择函数 f 计算选择的 $4P_i$ 的位。若差分轨迹有明显的峰值，则敌手就得出 $k_{t-2} = 0$；否则，$k_{t-2} = 1$。一旦确定了 k_{t-2}，敌手就能够类似地推出 k_{t-3}，依次类推。

对策 同 SPA 攻击一样，目前提出了有很多种抵抗 DPA 攻击的技术。同样，没有哪个提出的对策能够保证提供充足的保护，其有效性必须基于各种情况具体评估。本章的注释

一节中审视了这些对策。这里我们只给出一种对策，以抵抗针对点乘的上述特殊 DPA 攻击。

假设域$\mathbb{F}_q$的特征大于 3，并且在算法 5.7 中使用混合雅可比-仿射坐标（见 3.2.2 节）。于是点 P存储在仿射坐标中，点Q_0和Q_1存储在雅可比坐标中。首先将Q_1设为$Q_1 \leftarrow P$；若 $P=(x,y)$在仿射坐标中，则$Q_1=(x:y:1)$在雅可比坐标中。经过第一次设置后，Q_1的坐标随机设置为$(\lambda^2 x, \lambda^3 y, \lambda)$，其中$\lambda$是$\mathbb{F}_q$上随机选择的非零元素，算法同前面一样继续。上面描述的 DPA 攻击就被拦截，因为在随机的雅可比坐标下敌手不能预知$4P_i$的任何特殊位。

5.3.2 电磁分析攻击

COMS 设备通过的电流也会引起电磁（EM）泄露。EM 信号能够通过在设备附近放置传感器来采集。同能量分析攻击一样，人们现在能够分析 EM 信号来暴露正在执行的指令和寄存器中数据内容的信息。可以进行简单电磁分析（SEMA）攻击和差分电磁分析（DEMA）攻击，分别对应于 SPA 和 DPA 攻击的类似情况。同能量攻击一样，这些电磁分析（EMA）攻击是非闯入的，能够使用相对廉价的设备来执行。

因为 EM 泄漏取决于激活的门的物理特征，所以单独的 EM 传感器能捕捉不同类型的多个 EM 信号。这些信号能够被分离和独立分析。这与能量分析攻击中测量出的能量消耗是所有活动单元的能量消耗的集合不同。因此，EMA 攻击能够潜在地获得比能量分析攻击更多的信息，从而构成更大的威胁。

对于 EMA 攻击最深入的研究是在 2002 年由 IBM 研究人员 Agrawal, Archambeault, Rao 和 Rohatgi 承担的，他们在一些智能卡和包括 SSL 加速器的服务器上进行了实验。他们的实验表明，单独的宽带 EM 传感器的输出包含多种 EM 信号，每一种信号都能编码成关于设备状态的不同信息。此外他们还成功地使用 EMA 攻击了一些商用密码设备的安全性，这些设备尽管已经具有抵抗能量攻击的对策，但这也说明 EMA 攻击确实比能量攻击威胁更大。

与能量分析一样，对于 EMA 的对策应该是基于硬件的（如通过在 EM 辐射外包装金属屏蔽层或重新设计电路，来减少 EM 泄漏）或基于软件的（例如使用随机化的方法）。EMA 攻击的研究相对较新，哪种对策最有效还有待于观察。

5.3.3 错误消息分析

敌手能够利用的另外一个旁门是被攻击的密码设备产生的错误消息列表。例如，考虑公钥加密方案如 ECIES（见 4.5.1 节）的解密过程。因为在解密过程中遇到的一些数据条目没有写成要求的格式，密文可能被认为是无效的而被拒绝。对于 ECIES 解密（算法 4.43），

若对于 R 进行的嵌入的公钥确认失败，或者 $Z = hdR = \infty$，或者认证标志 t 无效的话，则密文 (R, C, t) 就会被拒绝。敌手获取拒绝的原因有多种途径。例如敌手可以访问错误日志文件，或者敌手能够准确地知道解密过程的时间，然后精确获取失败的位置。敌手获得拒绝的原因后，这些信息将更有利于攻击者的进一步分析。

为了展示这种类型的旁门攻击，我们考虑针对于 RSA-OAEP 加密方案的 Manger 攻击。尽管 RSA-OAEP 已经被证明是安全的（在随机预言模型下），Manger 攻击还是非常有效的。这也说明在传统安全模型中安全的密码方案，在现实世界中使用时并不一定安全。

RSA-OAEP 加密方案

RSA-OAEP 的目的是安全地传输短消息如对称会话密钥。它首先使用优化的非对称加密填充（OAEP）来将明文消息格式化，然后使用基本的 RSA 函数加密格式化后的消息。RSA-OAEP 在假定模 n 的 e 次根的问题是难解的并且使用的杂凑函数是随机函数的前提下，已被证明是安全的（以定义 4.41 的理解）。下面的符号用来描述加密和解密过程。

1. A 的 RSA 公钥是 (n, e)，d 是对应的私钥。整数 n 的长度为 k 字节。例如，若 n 是 1024 位的模数，则 $k = 128$。
2. H 是杂凑函数，输出长度是 l 字节。例如，若 H 是 SHA-1，则 $l = 20$。
3. P 包含一些编码参数。
4. 填充包括若干（可能为空）00 的字节串，后面跟一个 01 字节。
5. G 是掩码生成函数。它输入 s 字节串和输出长度 t，产生长度为 t 字节的（伪随机）字节串。实践中可以将 $G(s, t)$ 定义为对于 $0 \leqslant i \leqslant \lceil t/l \rceil - 1$ 连续的杂凑值 $H(s \| i)$ 的级联，并根据需要删去掉右边的若干字节。

S 掩码后的 $maskedS$ 和 PM 掩码后的 $maskedPM$ 的级联 $\overline{m}$ 是长度为 $k-1$ 的字节串。这保证了 $\overline{m}$ 的整数表示 m 小于长度是 k 字节的模数 n，从而 m 能从 c 中恢复。

算法 5.8 RSA-OAEP 加密

输入：RSA 公钥 (n, e)，长度最多是 $k-2-2l$ 字节的消息 M。

输出：密文 c。

1. 选择长度为 l 字节的随机种子 S。
2. 执行图 5.15 描述的 OAEP 编码操作，输入为 S，P 和 M，得到整数 m：
 2.1 通过级联 $H(P)$、合适长度的填充串和 M，得到填充后的长度为 $k-l-1$ 字节的消息 PM。

2.2 计算$maskedPM = PM \oplus G(S, k-l-1)$。

2.3 计算$maskedS = S \oplus G(maskedPM, l)$。

2.4 将$maskedS$和$maskedPM$级联，并将结果$\overline{m}$转换为整数m。

3. 计算$c = m^e \bmod n$。

4. 返回(c)。

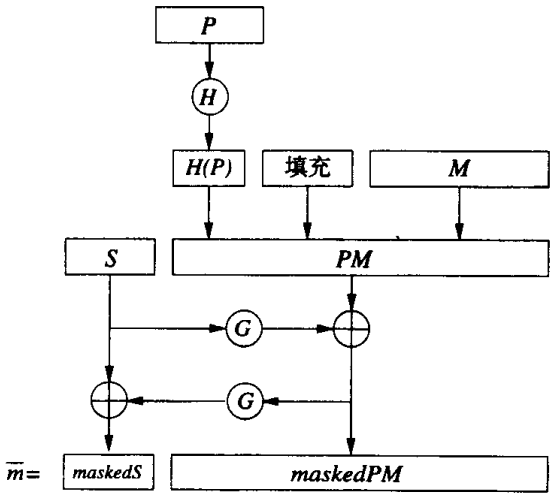


图 5.15 OAEP 编码函数

算法 5.9 RSA-OAEP 解密

输入：RSA 公钥(n, e)，私钥d，密文c。

输出：明文M或者拒绝该密文。

1. 检查是否$c \in [0, n-1]$；若不是，则返回（“拒绝该密文”）。

2. 计算$m = c^d \bmod n$。

3. 将m转换为长度为k的字节串$\overline{m}$。令X表示$\overline{m}$的第一个字节。

4. 若$X \neq 00$，则返回（“拒绝该密文”）。

5. 执行图 5.16 描述的 OAEP 解码操作，输入为P和$\overline{m}$：

5.1 分解$\overline{m}$得到X，一个长度为l的字节串$maskedS$，和一个长度为$k-l-1$的字节串$maskedPM$。

5.2 计算$S = maskedS \oplus G(maskedPM, l)$。

5.3 计算$PM = maskedPM \oplus G(S, k-l-1)$。

5.4 将PM分成一个由PM的前l个字节组成的字节串Q，一个由Q后面所有连续的零（可能为空）组成的字节串PS，一个字节T和一个字节串M。

5.5 若$T \neq 01$，则返回（“拒绝该密文”）。

5.6 若$Q \neq H(P)$，则返回（“拒绝该密文”）。

6. 返回(M)。

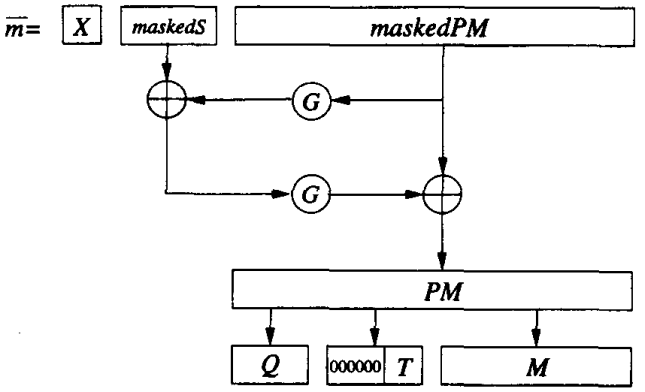


图 5.16 OAEP 解码函数

Manger 攻击和对策

密文$c' \in [0, n-1]$无效的原因可能有几个：或者是算法 5.9 的步骤 4 中$X \neq 00$，或者是步骤 5.5 中$T \neq 01$，或者是步骤 5.6 中$Q \neq H(P)$。Manger 攻击假设当c'被解密器认定无效时敌手能够确定是否$X \neq 00$。攻击不需要选择密文攻击的全部权力——敌手不需要掌握所选择密文的对应明文。

假设现在敌手想要解密由 A 的 RSA 密钥加密的目标密文 c。因为 c 是合法的，敌手预先知道$m = c^d \bmod n$在区间$I = [0, 2^{8(k-1)} - 1]$内。敌手选择与 c 相关的密文c'，m'的最左字节X'是否满足$X' \neq 00$的信息使得敌手将包含明文 m 的区间 I 的范围缩小（大约）一半。我们这里将不给出如何选择c'的技术细节，但要指出的是，选择可以高效地进行。通过提供 $8k$ 个这样的密文c'给 A，并掌握哪里有对应的X'满足$X' \neq 00$，区间 I 将缩小到只有一个整数，于是敌手就解出了 m 并能够容易地算出明文 M。若 n 是 1024 位的整数，则只需要与被攻击者交互 1024 次，因此这种攻击方法非常实用。

通过保证解密过程中三种检查失败时返回相同的错误消息，可防止该攻击。此外，为了防止敌手通过解密操作的时间推断出错误发生的位置，算法 5.9 的步骤 4 和步骤 5.5 的检查应该推迟到计算出$H(P)$并与 Q 相比较的步骤 5.6 中同时进行。

5.3.4 故障分析攻击

Boneh，DeMillo 和 Lipton 发现，若一个密码设备执行私钥操作时发生错误，则密码操

作的输出可能不正确，从而会向敌手提供有用的信息。这样的错误可能由非恶意的事件产生（如硬件故障、软件缺陷或外部噪声），也可能由能够接近物理设备的敌手有意地产生。

故障分析攻击通常在实践中不构成显著威胁。然而如果密码操作的环境容易产生非恶意或者恶意的错误，那么就应该采取合理的防范措施。这包括对于计算的结果在提交前的验证，在内部存储器中使用错误控制技术来检测或纠正数据错误。

下面我们给出在 RSA 签名方案中的故障分析攻击和相应的对策。

RSA 签名的生成

考虑 RSA 签名方案的 FDH（全域杂凑）变体，公钥(n,e)和私钥 d。消息 M 的签名是

$$s = m^d \bmod n, \tag{5.7}$$

其中 $m = H(M)$，H 是杂凑函数，输出是区间$[0, n-1]$内的整数。M 的签名 s 的验证是通过计算$m = H(M)$和$m' = s^e \bmod$，再检查$m = m'$来完成的。

为了加速签名操作(5.7)，签名者计算

$$s_p = m^{d_p} \bmod p \text{和} s_q = m^{d_q} \bmod q, \tag{5.8}$$

其中 p 和 q 是 n 的素因子，$d_p = d \bmod (p-1)$，$d_q = d \bmod (q-1)$。签名 s 的计算可以通过 $s = as_p + bs_q \bmod n$ 来完成，其中 a 和 b 是满足下式的整数：

$$a \equiv \begin{cases} 1 & (\bmod\ p) \\ 0 & (\bmod\ q) \end{cases} \text{和} \quad b \equiv \begin{cases} 0 & (\bmod\ p) \\ 1 & (\bmod\ q) \end{cases}。$$

签名者可以预先计算d_p，d_q，a 和 b。这样的签名过程会更快，因为在式(5.8)中的两个乘方取模运算的指数和模数的长度是式(5.7)中的指数和模数的一半。

假设现在计算s_p时出现错误，而计算s_q没有错误。特别地，假设$s_p \neq m^{d_p} \pmod p$和$s_q \equiv m^{d_q} \pmod q$。于是有

$$s \not\equiv m^{d_p} \pmod p \text{和} s \equiv m^{d_q} \pmod q,$$

由此有

$$s^e \not\equiv m \pmod p \text{和} s^e \equiv m \pmod q。$$

由此得出

$$\gcd(s^e - m, n) = q, \tag{5.9}$$

于是获得消息表示 m 和(不正确的)签名 s 的敌手就能够容易地分解 n，从而计算出私钥 d。

对于 RSA 签名的特殊故障分析攻击的防范方法之一是，从消息 M 转到消息表示 m 的形式时加入一些随机信息，这样敌手就不能从错误的签名中得出 m(也就计算不出来式(5.9)中的 gcd)。这一点在 RSA 签名方案的变体 PSS(概率签名方案)中满足。然而要注意可能存在其他类型的对于 PSS 有效的故障分析攻击。

最简单和最有效的对策是在传送前加强设备对于签名的检验。

5.3.5 时间分析攻击

时间分析攻击的前提是执行算术操作的时间基于操作数的具体值而改变。敌手若能够精确测量设备执行密码操作所用的时间(例如智能卡上签名的生成)，就能够分析测量的结果从而推断出关于密钥的信息。时间攻击与能量攻击相比，通常对诸如智能卡等设备的威胁不大，因为需要进行很大数量的测量。然而 Boneh 和 Brumley 最近的研究表明，在工作站上通过本地网络运行带 RSA 的 SSL(这时能量攻击不可行)，可能会受到时间攻击。因而，为谨慎起见，安全工程师应该考虑让他们的系统能够抵抗时间攻击。

已经有文献给出对于 RSA 和 DES 实现的时间攻击的实验结果，但是还没有对于椭圆曲线系统实现的时间攻击的公开报道。预期对于椭圆曲线签名方案如 ECDSA 的攻击会特别困难，因为当每次调用签名生成程序时都会选择一个新的每条消息的秘密 k。

5.4 注释和进一步参考

5.1 节

Intel IA-32 处理器系列的特性在[210]中给出。关于 Pentium 处理器系列优化的参考文献包括 Intel 手册[208，209]和 Gerber[171]。[4]中给出了 AMD K6 处理器的 SIMD 能力的细节。关于指令的延迟和吞吐量的脚注 1 源于 Intel[209]。

SPARC 规范由国际 SPARC 体系结构委员会(http://www.sparc.org)制定，Weaver 和 Germond[476]对其进行了文档化；也可参见 Paul[371]。V9 的设计被得州仪器、Sun SuperSPARC 和 Ross 技术 HyperSPARC 赶超，都支持超标量。例 5.1 和例 5.2 部分基于 GNU MP 版本 4.1.2。

5.1.2 节中描述的使用浮点操作在 P-224 上对于有限域和椭圆曲线运算的快速实现是由 Bernstein[42, 43]提出的。[42]还提供了发展历史信息和参考文献。针对 P-224 提出的方法的

许多分析在2002年还不完全。Bernstein宣称在他即将出版的书籍“High-speed Cryptography”中将包含有“在NIST P-224椭圆曲线上的快速点乘”。

SIMD经常在关于图像和语音的应用中使用，Intel[209]还建议将其用于“加密算法”。Aoki和Lipmaa[17]评估了MMX技术在AES候选算法上的效果，指出MMX对于Rijndael非常有效；也可参见Lipmaa[298]的IDEA分组密码的实现。在为了求解Certicom ECC2K-108挑战[88]（109位二进制域上Koblitz曲线的椭圆曲线离散对数问题的实例）而编写的跨平台代码中，Robert Harley[191]提供了域运算程序的不同版本。MMX版本的速度大约是只使用通用寄存器的版本的“两倍”。Karatsuba-style方法对于给定目标表现很好；然而在我们的测试中，算法2.36最快的版本只使用了通用寄存器。

例5.2的整数乘法在SSE2指令集中只使用了标量操作。Moore[332]利用128位SSE2寄存器的向量能力在两个64位的寄存器中来同时执行两个32位数值的乘法。方法是对操作数扫描，获得在大小为4×4的子阵（对应于128位寄存器对中的数值）中29位的值a_i和b_j的乘积矩阵（$a_i b_j$）。广泛使用了一个变换指令（pshufd）把从给定寄存器选出的四个32位组件载入到寄存器中。乘积进行累加，但“进位处理”在第二阶段进行。代码能够对不同大小的输入方便地改变；然而在例5.2中讨论的特殊情况下，该方法要比使用标量操作的（固定长度）的乘积扫描方法慢。

近年来，对于在普通通用寄存器上的实现细节和时间的研究中，Lim和Hwang[293, 294]的文章提供了广泛的测试数据（在Intel Pentium II和DEC Alpha上），特别是对于OEF。Smart[440]对于椭圆曲线方法比较了大概相同大小的有代表性的素域、二进制域和最佳扩域，给出了域运算和椭圆曲线操作在Sun UltraSPARC III和Intel Pentium Pro上的时间。编码是用C++和有限的几行嵌入的汇编语言编写的；在二进制域上计算次数少于8的多项式乘法中使用了Karatsuba-Ofman方法进行查表。Hankerson，López和Menezes[189]以及Brown，Hankerson，López和Menezes[77]对于软件实现NIST曲线进行了深入研究，给出了Intel Pentium II上域和曲线的运算时间。De Win，Mister，Preneel和Wiener[111]将ECDSA同DSA和RSA进行了比较。在Intel Pentium Pro上对于素域使用了有限的汇编语言；通过Barrett约减。Schroeppel，Orman，O'Malley和Spatscheck[415]给出了二进制域上的算法；特别是测试了两个约减三项式的殆逆算法（算法2.50）的时间，其中一个很适合使用殆逆方法。

对于在受限制的设备如智能卡和掌上设备上的实现，需要面对不同的挑战和目标。Naccache和M'Raïhi[339]对于智能卡的密码实现给出了介绍性的纵览。Durand[126]比较了素数特征的域上的求逆算法，给出了在SGS-Thomson的RISC处理器上RSA解密和椭圆曲

线点乘的运算时间。Hasegawa，Nakajima 和 Matsui[194]在三菱的 16 位 CISC M16C 上实现了 ECDSA。低的存储开销是至关重要的，椭圆曲线点的操作在他们的实现中只使用了两个临时变量。包含 SHA-1 在内的 ECDSA 的实现需要 4000 个字节。为了增加效率，使用了一个形式为$p = e2^a \pm 1$的素数，其中e刚好在一个字（16 位）内，a是字大小的倍数；特别地，在实现中使用了160位的$p = 65112 \cdot 2^{144} - 1$。Itoh，Takenaka，Torii，Temma 和 Kurihara[216]在得州仪器数字信号处理器 TMS320C620 上实现了 RSA，DSA 和 ECDSA。[260]讨论了 Montgomery 乘法算法的流水线改进。连续的加倍算法减少了域乘法的次数（在 Cohen，Miyaji 和 Ono[100]中使用了改进的雅可比坐标方法）；在假设除 2 的开销同域加法的开销相似的前提下，域加法也减少了（见 3.2.2 节）。Guajardo，Blümel，Krieger 和 Paar[182]基于得州仪器 MSP430x33x 系列 16 位 RISC 微控制器构建了低能量低开销设备。实现在素数$p = 2^{128} - 2^{97} - 1$的域$\mathbb{F}_p$上，适合于低安全要求的应用。求逆基于Fermat定理，取模中使用了特殊的形式，以减少在k列求幂方法的预计算数目。

在一些受限制的设备上适合使用 OEF。Chung，Sim 和 Lee[97]讨论了在带有 MAC2424 算术协处理器的低功耗三星 CalmRISC 8 位处理器上的性能和实现。协处理器可在 24 位或者 16 位模式下工作；由于性能的限制而选择 16 位模式。给出了在$p = 2^{16} - 165$和约减多项式$f(z) = z^{10} - 2$的域$\mathbb{F}_{p^{10}}$上的域运算和曲线操作的时间。Woodbury，Bailey 和 Paar[486]测试了在低功耗 Intel 8051 系列处理器上的点乘。可用的 RAM 只有 256 字节，同慢速的外部 XRAM 一起进行预计算。实现是在约减多项式$f(z) = 2^{17} - 2$的 OEF$\mathbb{F}_{(2^8-17)^{17}}$上的曲线上，适合于低安全要求的应用。

个人数字助理 PDA 如 Palm 和 RIM 与上面提到的受限制设备相比，提供了固化的存储器和处理能力，但是能力弱于普通便携式电脑，在功耗和通信带宽上存在限制。Weimerskirch，Paar 和 Chang Shantz[477]提供了在 2 MB 内存和 16 MHz Motorola Dragonball 的 Handspring Visor 上的实现结果。给出了 NIST 推荐的$\mathbb{F}_{2^{163}}$上的随机和 Koblitz 曲线的时间。

5.2 节

图 5.3 中描述的椭圆曲线处理器系统结构是由 Orlando 和 Paar[361]提出的。

Beth 和 Gollmann[45]描述了包括 MSB 和 LSB 版本的$\mathbb{F}_{2^m}$乘法器的几种电路，其中一种使用标准基和双基表示。串行数字乘法器（算法 5.5）由 Song 和 Parhi[449]提出。在$\mathbb{F}_{2^m}$上求逆的算法 5.6 由 Itoh 和 Tsujii[217]提出（也可参见 Agnew，Beth，Mullin 和 Vanstone[5]）。算法中在$\mathbb{F}_{2^m}$上的元素使用标准基表示。Guajardo 和 Paar[183]对于普通扩域（包括最佳扩域）上的求逆改编了算法，使用了多项式基表示。

描述椭圆曲线操作的硬件实现的文章有很多。这些文章的大多数考虑的是二进制域上的椭圆曲线。Orlando 和 Paar[361]提出了适于二进制域上椭圆曲线操作的 FPGA 实现的可扩展处理器系统结构。Song 和 Parhi[449]提出了使用串行数字电路执行乘法的方法。给出了域 $\mathbb{F}_{2^{167}}$上的执行时间。Okada，Torii，Itoh 和 Takenaka[353]描述了$\mathbb{F}_{2^{163}}$上椭圆曲线的 FPGA 实现。Bednara 等[32]（也可参见 Bednara 等[33]）比较了他们在域$\mathbb{F}_{2^{191}}$上的多项式基和标准基表示的椭圆曲线操作的 FPGA 实现。他们的结论是，多项式基乘法器比标准基乘法器的实现需要更少的逻辑门，Montgomery 方法（算法 3.40）更适于点乘。

Ernst，Jung，Madlener，Huss 和 Blümel[134]使用 Karatsuba-Ofman 方法进行了二元多项式相乘。Goodman 和 Chandrakasan[177]，Schroeppel，Beaver，Gonzales，Miller 和 Draelos[414]考虑了倾向于能量消耗最小化的硬件设计。Gura 等[186]设计了硬件加速器，允许使用 $m \leqslant 255$ 的任意二进制域$\mathbb{F}_{2^m}$上的任意椭圆曲线。Paar 和 Soria-Rodriguez[365]研究了使用二进制域的子域的系统结构。

Agnew，Mullin，Onyszchuk 和 Vanstone[6]（对于域$\mathbb{F}_{2^{593}}$），Agnew，Mullin 和 Vanstone[7]（对于域$\mathbb{F}_{2^{155}}$），Gao，Shrivastava 和 Sobelman[162]（对于任意二进制域），Leong 和 Leung[286]（对于域$\mathbb{F}_{2^{113}}$，$\mathbb{F}_{2^{155}}$和$\mathbb{F}_{2^{173}}$），分别描述了使用标准基表示的二进制域运算的硬件实现。后两篇文章包括有限域操作和椭圆曲线操作。

Koren 的书[266]是对执行加法、减法和乘法的基本整数操作的硬件系统结构的极好介绍。Orlando 和 Paar[362]详细描述了在素域上执行椭圆曲线运算的可扩展硬件系统结构。

Savaş，Tenca 和 Koç[404]以及 Großschädl[181]介绍了在素域和二进制域上执行乘法的可扩展乘法器。对于两种设计，统一的乘法器与素域的单独乘法器相比，只需要增加很少的面积。Savaş，Tenca 和 Koç 设计的乘法是使用 Montgomery 技术（见 2.2.4 节）执行的，而 Großschädl 的设计使用的是累加部分乘积的更常规方法。Gutub，Tenca，Savaş 和 Koç[187]研究了在素域和二进制域上 Montgomery 求逆的统一设计。Wolkerstorfer[485]提出了执行二进制域和素域上所有操作的低功耗系统结构。

Bertoni 等[44]提出了执行 p 为素数的$\mathbb{F}_{p^m}$上的乘法的硬件系统结构，特别是$p = 3$的情况；也可参见 Page 和 Smart[366]。

5.3 节

自 1999 年开始每年举办一次的“Cryptographic Hardware and Embedded Systems”会议给出了大量关于旁门攻击和相应对策的研究报告。这些会议的论文集由 Springer-Verlag 出版社出版[262，263，261，238]。旁门攻击不包括通常的编程和操作错误，如缓冲溢出、

可预测的随机数生成器、紊乱情况和口令选择不当等。若要了解这些错误的安全性意义，可参阅 Anderson[11]以及 Viega 和 McGraw[473]的书籍。

SPA 和 DPA 攻击在 1998 年由 Kocher，Jaffe 和 Jun[265]提出。Coron[104]首先将这些攻击应用到椭圆曲线密码方案，并对点乘提出了抵抗 SPA 的方法（算法 5.7），以及抵抗 DPA 的随机投影坐标方法。Oswald[364]展示了如何利用执行二元 NAF 点乘方法（算法 3.31）时的能量轨迹得出的关于 NAF(k)的部分信息，来确定乘数 k。Akkar，Bevan，Dischamp 和 Moyart[9]以及 Messerges，Dabbish 和 Sloan[323]报告了在智能卡上能量分析攻击的实验结果，而 Gebotys 和 Gebotys[168]报告了在 DSP 处理器内核上的实验结果。图 5.13 和图 5.14 取自 Gebotys 和 Gebotys[168]。

Chari，Jutla，Rao 和 Rohatgi[91]给出了一些普通 SPA 和 DPA 的对策，并给出了评估它们的效果的形式化方法。关于防范能量分析攻击的建议，基于硬件的防护措施包括使用内置电源，随机化被执行的指令的次序（May，Muller 和 Smart[308]），随机化重命名的寄存器（May，Muller 和 Smart[309]），使用两个电容，一个通过外部电源充电，另一个给设备供应能量（Shamir[422]）。

一种防范 SPA 攻击点乘的有效方法是使用椭圆曲线加法公式来进行倍点运算。Liardet 和 Smart[291]研究了对于雅可比形式的曲线的这种方法，Joye 和 Quisquater[231]研究了 Hessian 形式的方法。Izu 和 Takagi[221]设计了针对 Brier-Joye 公式的主动攻击（未使用能量分析），使用混合乘数进行点乘的椭圆曲线方案能够恢复私钥的几个比特。抵抗 SPA 的另外一种策略是使用特殊的点乘算法，如 Coron 算法（算法 5.7），加法和倍点操作与乘数无关。另外的方法是 Montgomery 点乘（见算法 3.40，也可参见 Okeya 和 Sakurai[358]），Möller[327，328]，Hitchcock 和 Montague[198]以及 Izu 以及 Takagi[220]提出的方法。Möller [327]和 Izu-Takagi[220]方法（改进版本）的安全性和效率由 Izu，Möller 和 Takagi[219]进行了细致分析。Trichina 和 Bellezza[461]以及 Gebotys 和 Gebotys[168]使用另外一种策略，设计出了加法和倍点操作的公式，它具有相同的域操作数目（加法、减法、乘法和平方）。

Hasan[193]研究了 Koblitz 曲线（见 3.4 节）上的点乘的能量分析攻击，并提出了一些对于性能没有明显影响的对策。

Joye 和 Tymen[232]提出了对于给定的椭圆曲线使用随机选择的同构以及随机选择的域表示形式来抵抗 DPA 攻击的方法。Goubin[180]展示了即使点乘已经使用如算法 5.7 的抗 SPA 的方法，以及如随机化投影坐标、随机化椭圆曲线或者随机化域表示的抗 DPA 的方法，点乘仍然有可能在敌手能够选择基点（如 ECIES 就会出现这种情况）的情况下受到 DPA 的攻击。Goubin 的研究突出了对于能量分析攻击设计安全的点乘方法的困难性。

电磁辐射泄漏的可能在军事上很早就被关注。例如在美国国家安全局[343]最近不再当机密文件处理的 TEMPEST 文档中，调查研究了不同辐射的威胁，包括电磁辐射、线路架设、声音散发等。关于攻击技术和对策的不保密的文献也非常多。例如 Kuhn 和 Anderson[272]讨论了基于软件的发起和阻止攻击的技术，基于在显示屏幕电磁辐射散发的信息的推导。Loughry 和 Umphress[302]描述了如何对计算机 LED（发光二极管）状态指示器的光辐射散发进行分析，推断出设备正在处理的数据。Anderson 的书[11]中的第 15 章对放射安全性提供了很好的介绍。在密码设备如智能卡上的电磁分析（EMA）攻击的实验结果以及与能量分析攻击的比较，首先由 Quisquater 和 Samyde[386]以及 Gandolfi，Mourtel 和 Olivier[161]给出。目前公开的对于 EMA 攻击的研究中最全面的是 Agrawal，Archambeault，Rao 和 Rohatgi[8]的工作。

利用错误消息的旁门攻击的第一个突出实例是 1998 年 Bleichenbacher[53]对于 PKCS#1 v1.5 标准的 RSA 加密方案的攻击。该版本在使用 RSA 函数之前对于明文消息进行格式化，该版本的 RSA 加密在包括提供安全网络通信的 SSL 协议在内的实践中被广泛使用。对于 1024 位的 RSA 模空间，通过提交大约 100 万个精心选择的与 c 相关的密文给被攻击者，并由此获知密文是否被拒绝，Bleichenbacher 攻击使敌手能够获得目标密文 c 的解密。该攻击使得无数 SSL 实现需要打补丁。RSA-OAEP 加密方案由 Bellare 和 Rogaway[38]提出，Shoup[427]以及 Fujisaki，Okamoto，Pointcheval 和 Stern[153]在随机预言器模型下证明了该方案的安全性。这已经加入到了很多标准中，包括 PKCS#1[395] v2.2。Manger[303]在 2001 年提出了对于 RSA-OAEP 的攻击。Vaudenay[466]描述了当消息首先被填充和格式化后以 CBC 模式使用分组密码加密，对于对称密钥加密的错误消息分析攻击。

故障分析攻击在 1997 年首先由 Boneh，DeMillo 和 Lipton[56，57]研究，他们描述了这种方法对于 RSA 签名方案以及 Fiat-Shamir 和 Schnorr 身份证明协议的攻击。Bao 等[28]给出了针对 ElGamal，Schnorr 和 DSA 签名方案的故障分析攻击。RSA 签名方案的变体 FDH 和 PSS 由 Bellare 和 Rogaway[39]提出，他们在假设寻找模 n 的 e 次方根是难解的并且使用的杂凑函数是随机函数的前提下，证明了它们的安全性（在定义 4.28 的意义下）。针对椭圆曲线公钥加密方案的故障分析攻击由 Biehl，Meyer 和 Müller[46]提出，若在解密过程中由于一个错误而产生出不在合法椭圆曲线上的点，则他们的攻击就能成功。若能确保密码计算出来的点确实在正确的椭圆曲线上，则能够防止该攻击。Biham 和 Shamir[48]提出了针对 DES 对称密钥加密方案的故障分析攻击。Anderson 和 Kuhn[12]讨论了一些导致瞬间故障的现实方法，他们称之为短时脉冲波形干扰。最近，Skorobogatov 和 Anderson[437]展示了通过特殊的发光晶体管来导致智能卡故障的廉价设备；他们还提出了针对这些光学故

障感应攻击的相应对策。

1996 年，Kocher[264]介绍了时间分析攻击，他描述了针对 RSA 乘方取模的攻击。Schindler[407]给出了针对使用中国剩余定理的 RSA 指数运算实现的时间攻击。在智能卡上实现 RSA 的实验结果由 Dhem 等[117]报道。Hevia 和 Kiwi[197]描述了针对 DES 的时间攻击，得出了密钥的汉明重量。Brumley 和 Boneh[78]展示了时间攻击能够通过局域网从基于 OpenSSL 的 Web 服务器上窃取 RSA 私钥。Canvel，Hiltgen，Vaudenay 和 Vuagnoux[86]设计出了针对 SSL 和 TLS 中使用的 CBC 模式加密方案的时间攻击；他们的攻击能够解密出使用过的密文，例如一段口令的加密结果。

附录A 参数举例

本附录给出适合密码学用途的椭圆曲线的参数组 $D=(q,\mathrm{FR},S,a,b,P,n,h)$，其中每个符号的具体含义可参见4.2节。A.1节将给出判定既约多项式的算法，该算法可生成用来表示有限域 $\mathbb{F}_{p^m}$ 上元素的约减多项式。A.1节还将给出包括ANSI X9.62和ANSI X9.63在内的一些规范中推荐的既约二进制多项式的列表，这些既约二进制多项式表示二进制域 $\mathbb{F}_{2^m}$ 上的元素。NIST在FIPS 186-2标准中推荐美国政府使用的15个椭圆曲线将在A.2节列出。

A.1 既约多项式

一个次数 $m\geqslant 1$ 的多项式 $f(z)=a_m z^m+\cdots+a_1 z+a_0\in\mathbb{F}_p[z]$，若 $f(z)$ 不能被分解成 $\mathbb{F}_p[z]$ 上次数小于 m 的多项式的乘积，则称其为既约多项式。因为 $f(z)$ 既约当且仅当 $a_m^{-1}f(z)$ 既约，所以我们只用考虑首一多项式（即第一项系数 $a_m=1$ 的多项式）。

对于任意素数 p 和整数 $m\geqslant 1$，在 $\mathbb{F}_p[z]$ 上至少存在一个次数为 m 的首一既约多项式。事实上，这样的多项式的准确个数为

$$N_p(m)=\frac{1}{m}\sum_{d|m}\mu(d)p^{m/d},$$

其中 d 取值为所有 m 的正因子，Möbius函数定义为

$$\mu(d)=\begin{cases}1, & \text{若 } d=1,\\ 0, & \text{若 } d \text{ 可被素数的平方整除},\\ (-1)^l, & \text{若 } d \text{ 是 } l \text{ 个不同的素数的乘积。}\end{cases}$$

已知

$$\frac{1}{2m}\leqslant\frac{N_p(m)}{p^m}\approx\frac{1}{m}。$$

因此，如果我们能够快速地判断$\mathbb{F}_p[z]$上的多项式是否可约，那么就能够通过重复随机选择$\mathbb{F}_p[z]$上次数为m的首一多项式，直到找到既约多项式——预计的尝试次数为m。

算法A.1是判定可约性的快速算法。该算法是由$\mathbb{F}_p$上次数为m的多项式$f(z)$既约的充要条件是对于每个i（$1 \leqslant i \leqslant \lfloor \frac{m}{2} \rfloor$）都有$\gcd(f(z), z^{p^i} - z) = 1$而得出的。

算法A.1 判定一个多项式是否可约

输入：素数p和次数$m \geqslant 1$的多项式$f(z) \in \mathbb{F}_p[z]$。

输出：$f(z)$是否可约。

1. $u(z) \leftarrow z$。
2. 对于i从1到$\lfloor \frac{m}{2} \rfloor$，执行
 2.1 $u(z) \leftarrow u(z)^p \bmod f(z)$。
 2.2 $d(z) \leftarrow \gcd(f(z), u(z) - z)$。
 2.3 若$d(z) \neq 1$，则返回（“可约”）。
3. 返回（“既约”）。

对于每个m，$2 \leqslant m \leqslant 600$，表A.1和表A.2列出了$\mathbb{F}_2$上次数为$m$的既约三项式或五项式$f(z)$。表中标为“$T$”的列给出了多项式中除去首项$z^m$和常数项1之外非零项的次数。例如，$T = k$表示三项式$z^m + z^k + 1$，$T = (k_3, k_2, k_1)$表示五项式$z^m + z^{k_3} + z^{k_2} + z^{k_1} + 1$。ANSI X9.62和ANSI X9.63中给出了用来选择约减多项式的如下准则：

(i) 若存在$\mathbb{F}_2$上次数为m的既约三项式，则$f(z)$选取既约三项式$z^m + z^k + 1$中最小的k值。

(ii) 若不存在$\mathbb{F}_2$上次数为m的既约三项式，则$f(z)$选取既约五项式$z^m + z^{k_3} + z^{k_2} + z^{k_1} + 1$中(a)最小的$k_3$值；(b)$k_3$相同时，选取最小的$k_2$值；(c)$k_3$和$k_2$相同时，选取最小的$k_1$值。

表 A.1 次数为 m 的既约二进制多项式，$2 \leqslant m \leqslant 300$

m	T	m	T	m	T	m	T	m	T	m	T
1	–	51	6,3,1	101	7,6,1	151	3	201	14	251	7,4,2
2	1	52	3	102	29	152	6,3,2	202	55	252	15
3	1	53	6,2,1	103	9	153	1	203	8,7,1	253	46
4	1	54	9	104	4,3,1	154	15	204	27	254	7,2,1
5	2	55	7	105	4	155	62	205	9,5,2	255	52
6	1	56	7,4,2	106	15	156	9	206	10,9,5	256	10,5,2
7	1	57	4	107	9,7,4	157	6,5,2	207	43	257	12
8	4,3,1	58	19	108	17	158	8,6,5	208	9,3,1	258	71
9	1	59	7,4,2	109	5,4,2	159	31	209	6	259	10,6,2
10	3	60	1	110	33	160	5,3,2	210	7	260	15
11	2	61	5,2,1	111	10	161	18	211	11,10,8	261	7,6,4
12	3	62	29	112	5,4,3	162	27	212	105	262	9,8,4
13	4,3,1	63	1	113	9	163	7,6,3	213	6,5,2	263	93
14	5	64	4,3,1	114	5,3,2	164	10,8,7	214	73	264	9,6,2
15	1	65	18	115	8,7,5	165	9,8,3	215	23	265	42
16	5,3,1	66	3	116	4,2,1	166	37	216	7,3,1	266	47
17	3	67	5,2,1	117	5,2,1	167	6	217	45	267	8,6,3
18	3	68	9	118	33	168	15,3,2	218	11	268	25
19	5,2,1	69	6,5,2	119	8	169	34	219	8,4,1	269	7,6,1
20	3	70	5,3,1	120	4,3,1	170	11	220	7	270	53
21	2	71	6	121	18	171	6,5,2	221	8,6,2	271	58
22	1	72	10,9,3	122	6,2,1	172	1	222	5,4,2	272	9,3,2
23	5	73	25	123	2	173	8,5,2	223	33	273	23
24	4,3,1	74	35	124	19	174	13	224	9,8,3	274	67
25	3	75	6,3,1	125	7,6,5	175	6	225	32	275	11,10,9
26	4,3,1	76	21	126	21	176	11,3,2	226	10,7,3	276	63
27	5,2,1	77	6,5,2	127	1	177	8	227	10,9,4	277	12,6,3
28	1	78	6,5,3	128	7,2,1	178	31	228	113	278	5
29	2	79	9	129	5	179	4,2,1	229	10,4,1	279	5
30	1	80	9,4,2	130	3	180	3	230	8,7,6	280	9,5,2
31	3	81	4	131	8,3,2	181	7,6,1	231	26	281	93
32	7,3,2	82	8,3,1	132	17	182	81	232	9,4,2	282	35
33	10	83	7,4,2	133	9,8,2	183	56	233	74	283	12,7,5
34	7	84	5	134	57	184	9,8,7	234	31	284	53
35	2	85	8,2,1	135	11	185	24	235	9,6,1	285	10,7,5
36	9	86	21	136	5,3,2	186	11	236	5	286	69
37	6,4,1	87	13	137	21	187	7,6,5	237	7,4,1	287	71
38	6,5,1	88	7,6,2	138	8,7,1	188	6,5,2	238	73	288	11,10,1
39	4	89	38	139	8,5,3	189	6,5,2	239	36	289	21
40	5,4,3	90	27	140	15	190	8,7,6	240	8,5,3	290	5,3,2
41	3	91	8,5,1	141	10,4,1	191	9	241	70	291	12,11,5
42	7	92	21	142	21	192	7,2,1	242	95	292	37
43	6,4,3	93	2	143	5,3,2	193	15	243	8,5,1	293	11,6,1
44	5	94	21	144	7,4,2	194	87	244	111	294	33
45	4,3,1	95	11	145	52	195	8,3,2	245	6,4,1	295	48
46	1	96	10,9,6	146	71	196	3	246	11,2,1	296	7,3,2
47	5	97	6	147	14	197	9,4,2	247	82	297	5
48	5,3,2	98	11	148	27	198	9	248	15,14,10	298	11,8,4
49	9	99	6,3,1	149	10,9,7	199	34	249	35	299	11,6,4
50	4,3,2	100	15	150	53	200	5,3,2	250	103	300	5

表 A.2 次数为 m 的既约二进制多项式，$301 \leqslant m \leqslant 600$

m	T	m	T	m	T	m	T	m	T	m	T
301	9,5,2	351	34	401	152	451	16,10,1	501	5,4,2	551	135
302	41	352	13,11,6	402	171	452	6,5,4	502	8,5,4	552	19,16,9
303	1	353	69	403	9,8,5	453	15,6,4	503	3	553	39
304	11,2,1	354	99	404	65	454	8,6,1	504	15,14,6	554	10,8,7
305	102	355	6,5,1	405	13,8,2	455	38	505	156	555	10,9,4
306	7,3,1	356	10,9,7	406	141	456	18,9,6	506	23	556	153
307	8,4,2	357	11,10,2	407	71	457	16	507	13,6,3	557	7,6,5
308	15	358	57	408	5,3,2	458	203	508	9	558	73
309	10,6,4	359	68	409	87	459	12,5,2	509	8,7,3	559	34
310	93	360	5,3,2	410	10,4,3	460	19	510	69	560	11,9,6
311	7,5,3	361	7,4,1	411	12,10,3	461	7,6,1	511	10	561	71
312	9,7,4	362	63	412	147	462	73	512	8,5,2	562	11,4,2
313	79	363	8,5,3	413	10,7,6	463	93	513	26	563	14,7,3
314	15	364	9	414	13	464	19,18,13	514	67	564	163
315	10,9,1	365	9,6,5	415	102	465	31	515	14,7,4	565	11,6,1
316	63	366	29	416	9,5,2	466	14,11,6	516	21	566	153
317	7,4,2	367	21	417	107	467	11,6,1	517	12,10,2	567	28
318	45	368	7,3,2	418	199	468	27	518	33	568	15,7,6
319	36	369	91	419	15,5,4	469	9,5,2	519	79	569	77
320	4,3,1	370	139	420	7	470	9	520	15,11,2	570	67
321	31	371	8,3,2	421	5,4,2	471	1	521	32	571	10,5,2
322	67	372	111	422	149	472	11,3,2	522	39	572	12,8,1
323	10,3,1	373	8,7,2	423	25	473	200	523	13,6,2	573	10,6,4
324	51	374	8,6,5	424	9,7,2	474	191	524	167	574	13
325	10,5,2	375	16	425	12	475	9,8,4	525	6,4,1	575	146
326	10,3,1	376	8,7,5	426	63	476	9	526	97	576	13,4,3
327	34	377	41	427	11,6,5	477	16,15,7	527	47	577	25
328	8,3,1	378	43	428	105	478	121	528	11,6,2	578	23,22,16
329	50	379	10,8,5	429	10,8,7	479	104	529	42	579	12,9,7
330	99	380	47	430	14,6,1	480	15,9,6	530	10,7,3	580	237
331	10,6,2	381	5,2,1	431	120	481	138	531	10,5,4	581	13,7,6
332	89	382	81	432	13,4,3	482	9,6,5	532	1	582	85
333	2	383	90	433	33	483	9,6,4	533	4,3,2	583	130
334	5,2,1	384	12,3,2	434	12,11,5	484	105	534	161	584	14,13,3
335	10,7,2	385	6	435	12,9,5	485	17,16,6	535	8,6,2	585	88
336	7,4,1	386	83	436	165	486	81	536	7,5,3	586	7,5,2
337	55	387	8,7,1	437	6,2,1	487	94	537	94	587	11,6,1
338	4,3,1	388	159	438	65	488	4,3,1	538	195	588	35
339	16,10,7	389	10,9,5	439	49	489	83	539	10,5,4	589	10,4,3
340	45	390	9	440	4,3,1	490	219	540	9	590	93
341	10,8,6	391	28	441	7	491	11,6,3	541	13,10,4	591	9,6,4
342	125	392	13,10,6	442	7,5,2	492	7	542	8,6,1	592	13,6,3
343	75	393	7	443	10,6,1	493	10,5,3	543	16	593	86
344	7,2,1	394	135	444	81	494	17	544	8,3,1	594	19
345	22	395	11,6,5	445	7,6,4	495	76	545	122	595	9,2,1
346	63	396	25	446	105	496	16,5,2	546	8,2,1	596	273
347	11,10,3	397	12,7,6	447	73	497	78	547	13,7,4	597	14,12,9
348	103	398	7,6,2	448	11,6,4	498	155	548	10,5,3	598	7,6,1
349	6,5,2	399	26	449	134	499	11,6,5	549	16,4,3	599	30
350	53	400	5,3,2	450	47	500	27	550	193	600	9,5,2

A.2 椭圆曲线

在 FIPS 186-2 标准中，NIST 推荐了美国政府使用的 15 个不同安全级别的椭圆曲线。这些曲线分为三类：

(i) 素域$\mathbb{F}_p$上的随机椭圆曲线。

(ii) 二进制域$\mathbb{F}_{2^m}$上的随机椭圆曲线。

(iii) 二进制域$\mathbb{F}_{2^m}$上的 Koblitz 椭圆曲线。

曲线的参数将在 A.2.1 节、A.2.2 节和 A.2.3 节中分别给出。

在接下来的表中，整数和多项式通常被表示成十六进制串的形式。例如，“0x1BB5”是整数 7093 的十六进制表示。而二进制多项式$z^{13}+z^{11}+z^5+z^2+z+1$的系数构成的二进制串“10100000100111”用十六进制表示为“0x2827”。

A.2.1 $\mathbb{F}_p$上的随机椭圆曲线

表 A.3 列出了 NIST 推荐的五个在素域$\mathbb{F}_p$上随机选择的椭圆曲线的参数。素数 p 选择特殊的值，以便进行非常快速的模 p 约减（见 2.2.6 节）。在椭圆曲线方程中选择系数$a=-3$是为了对于用雅可比投影坐标表示的椭圆曲线的点进行加法时减少有限域上乘法的次数（见 3.2.2 节）。对于每个曲线，给出了下列参数：

p	素域$\mathbb{F}_p$的阶。
S	使用算法 4.17 随机生成椭圆曲线系数所需的种子。
r	算法 4.17 中 SHA-1 的输出。
a，b	椭圆曲线$y^2=x^3+ax+b$的系数，满足$rb^2\equiv a^3 \pmod p$。
n	基点 P 的（素数）阶。
h	余因子。
x，y	P 的 x 和 y 坐标。

表 A.3 NIST 推荐的素域上的随机椭圆曲线

P-192: $p = 2^{192} - 2^{64} - 1$, $a = -3$, $h = 1$

S = 0x 3045AE6F C8422F64 ED579528 D38120EA E12196D5

r = 0x 3099D2BB BFCB2538 542DCD5F B078B6EF 5F3D6FE2 C745DE65

b = 0x 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1

n = 0x FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831

x = 0x 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012

y = 0x 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

P-224: $p = 2^{224} - 2^{96} + 1$, $a = -3$, $h = 1$

S = 0x BD713447 99D5C7FC DC45B59F A3B9AB8F 6A948BC5

r = 0x 5B056C7E 11DD68F4 0469EE7F 3C7A7D74 F7D12111 6506D031 218291FB

b = 0x B4050A85 0C04B3AB F5413256 5044B0B7 D7BFD8BA 270B3943 2355FFB4

n = 0x FFFFFFFF FFFFFFFF FFFFFFFF FFFF16A2 E0B8F03E 13DD2945 5C5C2A3D

x = 0x B70E0CBD 6BB4BF7F 321390B9 4A03C1D3 56C21122 343280D6 115C1D21

y = 0x BD376388 B5F723FB 4C22DFE6 CD4375A0 5A074764 44D58199 85007E34

P-256: $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $a = -3$, $h = 1$

S = 0x C49D3608 86E70493 6A6678E1 139D26B7 819F7E90

r = 0x 7EFBA166 2985BE94 03CB055C 75D4F7E0 CE8D84A9 C5114ABC AF317768 0104FA0D

b = 0x 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B

n = 0x FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551

x = 0x 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296

y = 0x 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

P-384: $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, $a = -3$, $h = 1$

S = 0x A335926A A319A27A 1D00896A 6773A482 7ACDAC73

r = 0x 79D1E655 F868F02F FF48DCDE E14151DD B80643C1 406D0CA1 0DFE6FC5 2009540A
495E8042 EA5F744F 6E184667 CC722483

b = 0x B3312FA7 E23EE7E4 988E056B E3F82D19 181D9C6E FE814112 0314088F 5013875A
C656398D 8A2ED19D 2A85C8ED D3EC2AEF

n = 0x FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF C7634D81 F4372DDF
581A0DB2 48B0A77A ECEC196A CCC52973

x = 0x AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98 59F741E0 82542A38
5502F25D BF55296C 3A545E38 72760AB7

y = 0x 3617DE4A 96262C6F 5D9E98BF 9292DC29 F8F41DBD 289A147C E9DA3113 B5F0B8C0
0A60B1CE 1D7E819D 7A431D7C 90EA0E5F

P-521: $p = 2^{521} - 1$, $a = -3$, $h = 1$

S = 0x D09E8800 291CB853 96CC6717 393284AA A0DA64BA

r = 0x 000000B4 8BFA5F42 0A349495 39D2BDFC 264EEEEB 077688E4 4FBF0AD8 F6D0EDB3
7BD6B533 28100051 8E19F1B9 FFBE0FE9 ED8A3C22 00B8F875 E523868C 70C1E5BF
55BAD637

b = 0x 00000051 953EB961 8E1C9A1F 929A21A0 B68540EE A2DA725B 99B315F3 B8B48991
8EF109E1 56193951 EC7E937B 1652C0BD 3BB1BF07 3573DF88 3D2C34F1 EF451FD4
6B503F00

n = 0x 000001FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFA 51868783 BF2F966B 7FCC0148 F709A5D0 3BB5C9B8 899C47AE BB6FB71E
91386409

x = 0x 000000C6 858E06B7 0404E9CD 9E3ECB66 2395B442 9C648139 053FB521 F828AF60
6B4D3DBA A14B5E77 EFE75928 FE1DC127 A2FFA8DE 3348B3C1 856A429B F97E7E31
C2E5BD66

y = 0x 00000118 39296A78 9A3BC004 5C8A5FB4 2C7D1BD9 98F54449 579B4468 17AFBD17
273E662C 97EE7299 5EF42640 C550B901 3FAD0761 353C7086 A272C240 88BE9476
9FD16650

A.2.2 $\mathbb{F}_{2^m}$上的随机椭圆曲线

表 A.4 列出了 NIST 推荐的五个在二进制域 $\mathbb{F}_{2^m}$ 上随机选择的椭圆曲线的参数。扩展次数 m 为素数，以便 $\mathbb{F}_{2^m}$ 上存在几乎素数群阶的 Koblitz 曲线（见 A.2.3 节）。算法 4.19 用种子 S 在 $\mathbb{F}_{2^m}$ 上产生椭圆曲线的系数 b。算法的输出 b 为 $\mathbb{F}_{2^m}$ 上的元素，表示为高斯标准基，如 FIPS 186-2 所述。可以用换基矩阵将 b 转换为多项式表示的形式，详见 FIPS 186-2。对于每个曲线，给出了下列参数：

m	二进制域 $\mathbb{F}_{2^m}$ 的扩展次数。
$f(z)$	次数为 m 的约减多项式。
S	随机产生椭圆曲线系数所选的种子。
a，b	椭圆曲线 $y^2+xy=x^3+ax^2+b$ 的系数。
n	基点 P 的（素数）阶。
h	余因子。
x，y	P 的 x 和 y 坐标。

A.2.3 $\mathbb{F}_{2^m}$上的 Koblitz 椭圆曲线

表 A.5 列出了 NIST 推荐的五个二进制域上的 Koblitz 椭圆曲线的参数。二进制域 $\mathbb{F}_{2^m}$ 同 A.2.2 节中随机选择的椭圆曲线相同。选择 Koblitz 曲线的原因是点乘的速度比在随机曲线上快（见 3.4 节）。对于每个曲线，给出了下列参数：

m	二进制域 $\mathbb{F}_{2^m}$ 的扩展次数。
$f(z)$	次数为 m 的约减多项式。
a，b	椭圆曲线 $y^2+xy=x^3+ax^2+b$ 的系数。
n	基点 P 的（素数）阶。
h	余因子。
x，y	P 的 x 和 y 坐标。

表 A.4 NIST 推荐的二进制域上的随机椭圆曲线

B-163: $m=163$, $f(z)=z^{163}+z^7+z^6+z^3+1$, $a=1$, $h=2$

S = 0x 85E25BFE 5C86226C DB12016F 7553F9D0 E693A268

b = 0x 00000002 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD

n = 0x 00000004 00000000 00000000 000292FE 77E70C12 A4234C33

x = 0x 00000003 F0EBA162 86A2D57E A0991168 D4994637 E8343E36

y = 0x 00000000 D51FBC6C 71A0094F A2CDD545 B11C5C0C 797324F1

B-233: $m=233$, $f(z)=z^{233}+z^{74}+1$, $a=1$, $h=2$

S = 0x 74D59FF0 7F6B413D 0EA14B34 4B20A2DB 049B50C3

b = 0x 00000066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD

n = 0x 00000100 00000000 00000000 00000000 0013E974 E72F8A69 22031D26 03CFE0D7

x = 0x 000000FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36 F8F8EB73 71FD558B

y = 0x 00000100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA 36716F7E 01F81052

B-283: $m=283$, $f(z)=z^{283}+z^{12}+z^7+z^5+1$, $a=1$, $h=2$

S = 0x 77E2B073 70EB0F83 2A6DD5B6 2DFC88CD 06BB84BE

b = 0x 027B680A C8B8596D A5A4AF8A 19A0303F CA97FD76 45309FA2 A581485A F6263E31
3B79A2F5

n = 0x 03FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFEF90 399660FC 938A9016 5B042A7C
EFADB307

x = 0x 05F93925 8DB7DD90 E1934F8C 70B0DFEC 2EED25B8 557EAC9C 80E2E198 F8CDBECD
86B12053

y = 0x 03676854 FE24141C B98FE6D4 B20D02B4 516FF702 350EDDB0 826779C8 13F0DF45
BE8112F4

B-409: $m=409$, $f(z)=z^{409}+z^{87}+1$, $a=1$, $h=2$

S = 0x 4099B5A4 57F9D69F 79213D09 4C4BCD4D 4262210B

b = 0x 0021A5C2 C8EE9FEB 5C4B9A75 3B7B476B 7FD6422E F1F3DD67 4761FA99 D6AC27C8
A9A197B2 72822F6C D57A55AA 4F50AE31 7B13545F

n = 0x 01000000 00000000 00000000 00000000 00000000 00000000 000001E2 AAD6A612
F33307BE 5FA47C3C 9E052F83 8164CD37 D9A21173

x = 0x 015D4860 D088DDB3 496B0C60 64756260 441CDE4A F1771D4D B01FFE5B 34E59703
DC255A86 8A118051 5603AEAB 60794E54 BB7996A7

y = 0x 0061B1CF AB6BE5F3 2BBFA783 24ED106A 7636B9C5 A7BD198D 0158AA4F 5488D08F
38514F1F DF4B4F40 D2181B36 81C364BA 0273C706

B-571: $m=571$, $f(z)=z^{571}+z^{10}+z^5+z^2+1$, $a=1$, $h=2$

S = 0x 2aa058f7 3a0e33ab 486b0f61 0410c53a 7f132310

b = 0x 02F40E7E 2221F295 DE297117 B7F3D62F 5C6A97FF CB8CEFF1 CD6BA8CE 4A9A18AD
84FFABBD 8EFA5933 2BE7AD67 56A66E29 4AFD185A 78FF12AA 520E4DE7 39BACA0C
7FFEFF7F 2955727A

n = 0x 03FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF E661CE18 FF559873 08059B18 6823851E C7DD9CA1 161DE93D 5174D66E
8382E9BB 2FE84E47

x = 0x 0303001D 34B85629 6C16C0D4 0D3CD775 0A93D1D2 955FA80A A5F40FC8 DB7B2ABD
BDE53950 F4C0D293 CDD711A3 5B67FB14 99AE6003 8614F139 4ABFA3B4 C850D927
E1E7769C 8EEC2D19

y = 0x 037BF273 42DA639B 6DCCFFFE B73D69D7 8C6C27A6 009CBBCA 1980F853 3921E8A6
84423E43 BAB08A57 6291AF8F 461BB2A8 B3531D2F 0485C19B 16E2F151 6E23DD3C
1A4827AF 1B8AC15B

表 A.5 NIST 推荐的二进制域上的 Koblitz 曲线

K-163: $m=163$, $f(z)=z^{163}+z^7+z^6+z^3+1$, $a=1$, $b=1$, $h=2$ n = 0x 00000004 00000000 00000000 00020108 A2E0CC0D 99F8A5EF x = 0x 00000002 FE13C053 7BBC11AC AA07D793 DE4E6D5E 5C94EEE8 y = 0x 00000002 89070FB0 5D38FF58 321F2E80 0536D538 CCDAA3D9
K-233: $m=233$, $f(z)=z^{233}+z^{74}+1$, $a=0$, $b=1$, $h=4$ n = 0x 00000080 00000000 00000000 00000000 00069D5B B915BCD4 6EFB1AD5 F173ABDF x = 0x 00000172 32BA853A 7E731AF1 29F22FF4 149563A4 19C26BF5 0A4C9D6E EFAD6126 y = 0x 000001DB 537DECE8 19B7F70F 555A67C4 27A8CD9B F18AEB9B 56E0C110 56FAE6A3
K-283: $m=283$, $f(z)=z^{283}+z^{12}+z^7+z^5+1$, $a=0$, $b=1$, $h=4$ n = 0x 01FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFE9AE 2ED07577 265DFF7F 94451E06 1E163C61 x = 0x 0503213F 78CA4488 3F1A3B81 62F188E5 53CD265F 23C1567A 16876913 B0C2AC24 58492836 y = 0x 01CCDA38 0F1C9E31 8D90F95D 07E5426F E87E45C0 E8184698 E4596236 4E341161 77DD2259
K-409: $m=409$, $f(z)=z^{409}+z^{87}+1$, $a=0$, $b=1$, $h=4$ n = 0x 007FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFE5F 83B2D4EA 20400EC4 557D5ED3 E3E7CA5B 4B5C83B8 E01E5FCF x = 0x 0060F05F 658F49C1 AD3AB189 0F718421 0EFD0987 E307C84C 27ACCFB8 F9F67CC2 C460189E B5AAAA62 EE222EB1 B35540CF E9023746 y = 0x 01E36905 0B7C4E42 ACBA1DAC BF04299C 3460782F 918EA427 E6325165 E9EA10E3 DA5F6C42 E9C55215 AA9CA27A 5863EC48 D8E0286B
K-571: $m=571$, $f(z)=z^{571}+z^{10}+z^5+z^2+1$, $a=0$, $b=1$, $h=4$ n = 0x 02000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 131850E1 F19A63E4 B391A8DB 917F4138 B630D84B E5D63938 1E91DEB4 5CFE778F 637C1001 x = 0x 026EB7A8 59923FBC 82189631 F8103FE4 AC9CA297 0012D5D4 60248048 01841CA4 43709584 93B205E6 47DA304D B4CEB08C BBD1BA39 494776FB 988B4717 4DCA88C7 E2945283 A01C8972 y = 0x 0349DC80 7F4FBF37 4F4AEADE 3BCA9531 4DD58CEC 9F307A54 FFC61EFC 006D8A2C 9D4979C0 AC44AEA7 4FBEBBB9 F772AEDC B620B01A 7BA7AF1B 320430C8 591984F6 01CD4C14 3EF1C7A3

附录 B　ECC 标准

密码标准的重要性体现在两方面：(i)推进密码技术的广泛使用；(ii)促进不同实现环境的互操作性。通过完整详细地说明密码方案的每一步与诸如参数、密钥和交互消息等共享数据，以及限制实现者的可选设置，可以促进不同环境间的互操作性。

本节选择描述基于椭圆曲线机制的签名、加密和密钥建立的标准及其草案中的特点。表 B.1 中给出了这些标准的概要。这些标准的电子文档可以通过列在表 B.2 中的网址获得。需要注意的是，很多标准都在定时地更新，读者最好参考网站中的最新版本。

美国国家标准化组织（ANSI）　ANSI X9F 小组委员会隶属于 ANSI X9 委员会，该组织为业界制定信息安全标准。已出台有两个椭圆曲线标准：ANSI X9.62 规定 ECDSA（见 4.4.1 节），ANSI X9.63 规定各种椭圆曲线密钥的协定和密钥的传送协议，包括 STS（见 4.6.1 节）、ECMQV（见 4.6.2 节）和 ECIES（见 4.5.1 节）。这些标准的目标是达到高度的安全性和互操作性。基本的有限域限定为素域 $\mathbb{F}_p$ 或二进制域 $\mathbb{F}_{2^m}$。$\mathbb{F}_{2^m}$ 上的元素可以用 $\mathbb{F}_2$ 上的多项式基或者标准基表示。若使用多项式基，则约减多项式必须为不可约三项式，若不存在不可约三项式，则为不可约五项式。为了便于互操作，为每个有限域 $\mathbb{F}_{2^m}$ 推荐了一个特殊的约减多项式；这些多项式的次数为 m，$2 \leqslant m \leqslant 600$，在表 A.1 和表 A.2 中列出。若使用标准基，则要求用一个特殊的高斯标准基。主要的安全性要求是基点 P 的阶 n 必须大于 2^{160}。惟一使用到的杂凑函数是 SHA-1；ANSI X9.62 和 X9.63 将在下一年更新规范，以便允许使用更多的杂凑函数，从而输出不同长度的杂凑值。

美国国家标准技术研究所（NIST）　NIST 是美国政府商业部负责技术管理的联邦机构。其任务之一就是制定与安全相关的联邦信息处理标准（FIPS）供美国联邦政府部门使用。FIPS 标准被全世界广泛采用，其中包括数据加密标准（DES：FIPS 46），安全杂凑函数（SHA-1，SHA-256，SHA-384，SHA-512：FIPS 180-2[138]），高级加密标准（AES：FIPS 197[141]），以及基于杂凑的消息认证码（HMAC：FIPS 198[142]）。FIPS 186-2，即数字签名标准（DSS），规定了 RSA，DSA 和 ECDSA 数字签名方案。ECDSA 参见 ANSI X9.62，其推荐的 15 个椭圆曲线已在 A.2.1 节、A.2.2 节和 A.2.3 节列出。NIST 现在正在制定一套

椭圆曲线密钥建立方案的建议[342]，其中包括从 ANSI X9.63 中选取的协议。

电气和电子工程师协会（IEEE） IEEE P1363 工作组负责起草一系列公钥密码标准。P1363 涉及的范围很广，包括基于因数分解、基于有限域上的离散对数、基于椭圆曲线离散对数和基于格的难解问题的方案。1363-2000 标准包括椭圆曲线签名方案（ECDSA 以及由 Nyberg 和 Rueppel 提出的椭圆曲线对等签名方案），椭圆曲线密钥协商方案（ECMQV 和各种椭圆曲线 Diffie-Hellman（ECDH））。从根本上区别于 ANSI 标准和 FIPS 186-2 的地方是没有要求最小的安全需求而给出了广泛的选择空间。因此，其主要的目标是作为其他标准和应用可选择的各种密码协议规范的参考。1363-2000 标准中基本的有限域限定为素域$\mathbb{F}_p$或二进制域$\mathbb{F}_{2^m}$。P1363a 标准草案是1363-2000 的补充，包括ECIES 和提供数据恢复的 Pintsov-Vanstone 签名方案规范，并且允许使用包括最佳扩域（见 2.4 节）在内的奇数特征的扩域$\mathbb{F}_{p^m}$。

国际标准化组织（ISO） ISO 和国际电工委员会（IEC）在 SC 27 分组内部共同制定密码标准。ISO/IEC 15946 是一组椭圆曲线密码标准，规定了签名方案（包括 ECDSA 和 EC-KCDSA）、密钥建立方案（包括 ECMQV 和 STS）以及提供数据恢复的数字签名方案。ISO/IEC 18033-2 提供了包括 ECIES-KEM 和 PSEC-KEM 在内的各种公钥加密方案的详细描述和安全性分析。

密码标准化组织（SECG） SECG 是企业间为解决密码标准的互操作问题而成立的联盟。SEC 1 规定了 ECDSA，ECIES，ECDH 和 ECMQV，并尝试与所有 ANSI，NIST，IEEE 和 ISO/IEC 椭圆曲线标准兼容。在 SEC 2 中列出了包括 15 个 NIST 椭圆曲线在内的一些特殊椭圆曲线。

签名、完整性和加密的最新欧洲方案（NESSIE） NESSIE 工程由欧盟第五届程序计划所建立。其主要目标是评估和选择各种对称密码（分组密码、流密码、杂凑函数和消息认证码）和公钥密码（公钥加密、签名和认证方案）。椭圆曲线方案中被选取的是 ECDSA 及密钥传输协议 PSEC-KEM 和 ACE-KEM。

密码研究与评估委员会（CRYPTREC） 信息技术促进机构（IPA）在日本建立了 CRYPTREC 委员会，其目标是为日本政府评价电子商务中的密码协议。很多对称密码和公钥密码被评估，如 ECDSA，ECIES，PSEC-KEM 和 ECDH。

表 B.1　基于椭圆曲线的密码机制的标准及草案

标　准	年　份	名　称	参考文献
ANSI X9.62	1999	椭圆曲线数字签名算法	[14]
ANSI X9.63	2001	密钥协商和密钥传输	[15]
FIPS 186-2	2000	数字签名标准（DSS）	[140]
IEEE 1363-2000	2000	公钥密码规范标准	[204]
IEEE P1363a	（草案）	补充 1：附加的技术	[203]
ISO/IEC 15946-1	2002	基于椭圆曲线的技术——部分一：概述	[211]
ISO/IEC 15946-2	2002	部分二：数字签名	[212]
ISO/IEC 15946-3	2002	部分三：密钥建立	[213]
ISO/IEC 15946-4	（草案）	部分四：提供数据恢复的数字签名	[214]
ISO/IEC 18033-2	（草案）	加密算法——部分二：公钥密码	[215]
SEC 1	2000	椭圆曲线密码	[417]
SEC 2	2000	推荐的椭圆曲线参数	[418]

表 B.2　标准化组织的网址

ANSI	美国国家标准技术研究所，http://www.ansi.org
X9	业界商用安全标准，http://www.x9.org
IEEE	电气和电子工程师协会，http://www.ieee.org
P1363	公钥密码标准，http://grouper.ieee.org/groups/1363
ISO	国际标准化组织，http://www.iso.ch
IEC	国际电工委员会，http://www.iec.ch
SC27	信息技术—安全技术，http://www.din.de/ni/sc27
NIST	美国国家标准技术研究所，http://www.nist.gov
FIPS	联邦信息处理标准，http://www.itl.nist.gov/fipspubs
SECG	密码标准化组织，http://www.secg.org
SEC	密码标准文档，http://www.secg.org/secg_docs.htm
NESSIE	签名、完整性和加密的最新欧洲方案，http://www.cryptonessie.org
IPA	信息技术促进机构，http://www.ipa.go.jp/ipa-e/index-e.html
CRYPTREC	密码研究与评估委员会，http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

附录C 软件工具

本附录列出了一些对于实现和教学有用的工具，具体分为两节。C.1 节包含了针对研究的工具和其他工具，其中大部分都可以直接使用而不需要编程。C.2 节给出了一些特殊的库文件，需要使用编程语言例如 C 来调用。总之，C.1 节中的工具适用于教学，提供的是原型；而 C.2 节的内容适合程序开发者。研究人员可同时使用两节的工具包。对于这些工具包的介绍，大部分来自开发这些工具的作者本身。

C.1 通用工具

本节提供的工具在性能和界面上各不相同，以 bc 和 calc 作为基本开发工具，Maple，Mathematica 和 MuPAD 作为高级图形和用户界面开发工具。Magma 是比 Mathematica 更特别的工具，在诸如点运算等椭圆曲线运算方面能提供非常好的支持。本节还提到了 GAP 和 KANT/KASH 这两个特殊的工具包。

bc http://www.gnu.org

bc 是一种支持任意精度数之间运算的语言，在语法上与 C 语言有点相似。bc 的优点是其用途广泛和便于计算。Keith Matthews 用 bc 实现了很多关于数论的程序，http://www.numbertheory.org/gnubc/。

Calc http://www.gnu.org

Calc 提供易于进行数字计算的计算器功能。同时它能够通过编程实现复杂或冗长的计算。它提供了基本的模算术函数。Calc 由 David I. Bell 和 Landon Curt Noll 完成，http://sourceforge.net/projects/calc/。

GAP http://www.gap-system.org

GAP（Groups，Algorithms，Programming）是计算离散代数特别是计算群理论的系统。它可以进行长整数和有理数算术运算，包括分圆域、有限域、剩余环、多项式、向量和矩阵、

各种组合函数、初等数论以及其他各种运算。GAP 由 Lehrstuhl D für Mathematik 和 RWTH Aachen 于 1986 年在德国开发，后来于 1997 年被移植到了苏格兰的 St. Andrews 大学。

KANT/KASH http://www.math.tu-berlin.de/~kant/kash.html

计算代数数论的KANT 是专门用来进行数域和全体函数域运算的软件包。KASH 是 KANT 的前身 KAnt SHell 的缩写。KANT 是由 Dr. M. Pohst 教授在柏林大学开发的。

Magma http://magma.maths.usyd.edu.au

Magma 代数计算系统是“有良好支持的软件包，设计用来解决代数、数论、几何学和组合数学中的难解问题。它为代数、数论、几何学和组合数学提供了严格的数学环境。”特别是它对于椭圆曲线运算有更进一步的支持。

Magma 由悉尼大学数学与统计学院的计算代数小组开发和发行。“尽管 Magma 是非商业系统，但我们应该通过该软件的发行和技术支持来弥补其开销”。

Maple http://www.maplesoft.com

Maple 是一种高级的数学问题的解决和编程环境。Waterloo 大学的符号计算小组首先开发了 Maple 符号计算技术。Maple 是商业系统——在历史上对于学生和学者的许可是免费的。

Mathematica http://www.wolfram.com

Mathematica 是集成快速、高精度数字和符号运算且易于使用的数据可视化和编程能力的通用计算系统。Mathematica 的开发者是 Wolfram Research，于 1987 年在 Stephen Wolfram 创建。

MuPAD http://www.mupad.de

MuPAD 是进行数字和符号运算的通用代数计算系统。用户可以看到库文件的源代码，易于实现他们自己的程序和数据结构，还可以直接动态链接 C/C++编译模块，以提高速度和灵活性。

MuPAD 是由 Paderborn 大学（德国）的 B. Fuchssteiner 教授领导的 MuPAD 研究小组开发的。使用许可是免费的。商业版本可以通过 SciFace 软件公司获得。关于 MuPAD 出版了几本书，包括平装本的 *MuPAD Tutorial*: *A version and platform independent introduction*，作者是 J. Gerhard, W. Oevel, F. Postel 和 S. Wehmeier，由 Springer-Verlag 出版社 2000 年出版。

C.2 库

与 C.1 节中给出的软件包相比，本节所提供的软件包更为特殊。例如，一些库需要使用诸如 C 或 C++语言来编程。

GNU MP 是最基本的支持任意精度算术计算程序的库，它在许多平台上由于其效率都被推荐使用。Crypto++提供密码用途的一系列扩展程序，有着一流的 C++结构。OpenSSL，MIRACL 和 cryptlib 有着相似的特点。OpenSSL 由 SSLeay 发展而来，在 Apache 服务器的应用和 OpenSSH 中被广泛使用。MIRACL 提供了可执行的椭圆曲线点的计算。

除了整数和多项式算术以外，LiDIA 和 NTL 还提供了数论算法。连同 PARI-GP 一起，这些工具对于研究人员都很有用处。

cryptlib http://www.cs.auckland.ac.nz/~pgut001/cryptlib

尽管没有包含椭圆曲线，Peter Gutmann 提供的 cryptlib 安全工具由于其对于加密、数字签名、密钥和证书管理、消息安全服务以及各种加密硬件的广泛支持而备受瞩目。特别地，cryptlib 重视诸如 SSH，SSL，S/MIME 和 PGP 等高层服务的易用性。很多程序来自 OpenSSL。该工具集可运行于多种平台，对开放源代码和商用有两套许可，并且提供有丰富的文档。

Crypto++ http://www.eskimo.com/~weidai/cryptlib.html

Crypto++是 Wei Dai 开发的开放 C++库，实现了各种密码，包括加密、消息认证码、单向杂凑函数、公钥密码系统和密钥协商方案。http://sourceforge.net/projects/cryptopp/。

GNU MP http://www.swox.com/gmp/

GMP 是任意精度算术计算程序的开放库，操作对象为带符号整数、有理数和浮点数。其主要设计目标在于提高速度。

Libgcrypt http://www.gnu.org/directory/security/libgcrypt.html

Libgcrypt 是基于 GnuPG（基于应用的 OpenPGP）代码的通用密码函数库。它提供了构建密码模块的函数，包括对称密码、杂凑函数、消息认证码、公钥算法、大整数（从 GNU MP 的代码得来）和随机数。

LiDIA http://www.informatik.tu-darmstadt.de/TI/LiDIA/

LiDIA 是进行数论计算的 C++库，提供了多倍精度数据结构的高度优化实现。特别是

该库包括因数分解和椭圆曲线点的计算算法。由 Darmstadt 大学（德国）的 LiDIA 小组开发。

MIRACL http://indigo.ie/~mscott/

MIRACL 是多倍精度整数和有理数算法的 C/C++库，支持对称密码和公钥密码，包括椭圆曲线和椭圆曲线点的计算。Shamus 软件公司（爱尔兰）提供使用许可。它“对于非盈利用途、教育或其他非商业用途是免费的”。

NTL：进行数论计算的库 http://www.shoup.net/ntl/

NTL 是高性能的便携式 C++库，为任意长度的整数，或者有限域上的向量、矩阵和多项式，任意精度的浮点运算提供数据结构和算法。该库还包括先进的基于格的约减实现。NTL 由 Victor Shoup 维护。

OpenSSL http://www.openssl.org

OpenSSL 工程是为了实现安全套接字协议（SSL v2/v3）和传输层安全协议（TLS v1）以及通用密码库而共同开发的一个健壮的、全特征的开放源代码工具。OpenSSL 基于 Eric A. Young 和 Tim J. Hudson 开发的 SSLeay 库。

PARI-GP http://www.parigp-home.de

PARI-GP 是计算机辅助数论工具包，包括一个 C 库和可编程交互式 gp 计算器。它最初由 Henri Cohen 领导的团队在 Bordeaux 开发，现在由 Paris-Sud Orsay 大学的 Karim Belabas 和许多志愿者共同维护。

参考文献

[1] M. ABDALLA, M. BELLARE, AND P. ROGAWAY. The oracle Diffie-Hellman assumptions and an analysis of DHIES. *Topics in Cryptology—CT-RSA 2001* (LNCS 2020) [338], 143–158, 2001.

[2] C. ADAMS AND S. LLOYD. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, 2nd edition, 2002.

[3] L. M. ADLEMAN AND M.-D. A. HUANG, editors. *Algorithmic Number Theory—ANTS-I*, volume 877 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[4] ADVANCED MICRO DEVICES. *AMD-K6 Processor Multimedia Technology*, 2000. Publication 20726, available from http://www.amd.com.

[5] G. AGNEW, T. BETH, R. MULLIN, AND S. VANSTONE. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6:3–13, 1993.

[6] G. AGNEW, R. MULLIN, I. ONYSZCHUK, AND S. VANSTONE. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3:63–79, 1991.

[7] G. AGNEW, R. MULLIN, AND S. VANSTONE. An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2155}$. *IEEE Journal on Selected Areas in Communications*, 11:804–813, 1993.

[8] D. AGRAWAL, B. ARCHAMBEAULT, J. RAO, AND P. ROHATGI. The EM side-channel(s). *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 29–45, 2002.

[9] M. AKKAR, R. BEVAN, P. DISCHAMP, AND D. MOYART. Power analysis, what is now possible... *Advances in Cryptology—ASIACRYPT 2000* (LNCS 1976) [355], 489–502, 2000.

[10] E. AL-DAOUD, R. MAHMOD, M. RUSHDAN, AND A. KILICMAN. A new addition formula for elliptic curves over $GF(2^n)$. *IEEE Transactions on Computers*, 51:972–975, 2002.

[11] R. ANDERSON. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[12] R. ANDERSON AND M. KUHN. Low cost attacks on tamper resistant devices. *Security Protocols 1997* (LNCS 1361) [95], 125–136, 1998.

[13] R. ANDERSON AND S. VAUDENAY. Minding your p's and q's. *Advances in Cryptology—ASIACRYPT '96* (LNCS 1163) [245], 26–35, 1996.

[14] ANSI X9.62. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standards Institute, 1999.

[15] ANSI X9.63. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography. American National Standards Institute, 2001.

[16] A. ANTIPA, D. BROWN, A. MENEZES, R. STRUIK, AND S. VANSTONE. Validation of elliptic curve public keys. *Public Key Cryptography—PKC 2003* (LNCS 2567) [116], 211–223, 2003.

[17] K. AOKI AND H. LIPMAA. Fast implementation of AES candidates. *Third AES Candidate Conference (AES3)*, 2000. Available from http://csrc.nist.gov/encryption/aes/round2/conf3/aes3conf.htm.

[18] S. ARITA. Weil descent of elliptic curves over finite fields of characteristic three. *Advances in Cryptology—ASIACRYPT 2000* (LNCS 1976) [355], 248–258, 2000.

[19] D. ASH, I. BLAKE, AND S. VANSTONE. Low complexity normal bases. *Discrete Applied Mathematics*, 25:191–210, 1989.

[20] A. ATKIN AND F. MORAIN. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–68, 1993.

[21] I. ATTALI AND T. JENSEN, editors. *Smart Card Programming and Security*, volume 2140 of *Lecture Notes in Computer Science*. International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Springer-Verlag, 2001.

[22] D. AUCSMITH, editor. *Information Hiding—IH '98*, volume 1525 of *Lecture Notes in Computer Science*. Second International Workshop, IH'98, Portland, Oregon, April 1998, Springer-Verlag, 1998.

[23] E. BACH AND J. SHALLIT. *Algorithmic Number Theory, Volume I: Efficient Algorithms*. MIT Press, 1996.

[24] H. BAIER AND J. BUCHMANN. Elliptic curves of prime order over optimal extension fields for use in cryptography. *Progress in Cryptology—INDOCRYPT 2001* (LNCS 2247) [367], 99–107, 2001.

[25] D. BAILEY AND C. PAAR. Optimal extension fields for fast arithmetic in public-key algorithms. *Advances in Cryptology—CRYPTO '98* (LNCS 1462) [270], 472–485, 1998.

[26] ———. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14:153–176, 2001.

[27] R. BALASUBRAMANIAN AND N. KOBLITZ. The improbability that an elliptic curve has subexponential discrete log problem under the Menezes-Okamoto-Vanstone algorithm. *Journal of Cryptology*, 11:141–145, 1998.

[28] F. BAO, H. DENG, Y. HAN, A. JENG, D. NARASIMHALU, AND T. NGAIR. Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. *Security Protocols 1997* (LNCS 1361) [95], 115–124, 1998.

[29] P. BARRETT. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. *Advances in Cryptology—CRYPTO '86* (LNCS 263) [351], 311–323, 1987.

[30] L. BATTEN AND J. SEBERRY, editors. *Information Security and Privacy 2002*, volume 2384 of *Lecture Notes in Computer Science*. 7th Australasian Conference, July 3-5, 2001, Melbourne, Australia, Springer-Verlag, 2002.

[31] S. BEAUREGARD. Circuit for Shor's algorithm using $2n + 3$ qubits. *Quantum Information and Computation*, 3:175–185, 2003.

[32] M. BEDNARA, M. DALDRUP, J. SHOKROLLAHI, J. TEICH, AND J. VON ZUR GATHEN. Reconfigurable implementation of elliptic curve crypto algorithms. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002. Available from http://computer.org/proceedings/ipdps/1573/workshops/1573toc.htm.

[33] ———. Tradeoff analysis of FPGA based elliptic curve cryptography. *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2002)*, 2002.

[34] M. BELLARE, editor. *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*. 20th Annual International Cryptology Conference, Santa Barbara, California, August 2000, Springer-Verlag, 2000.

[35] M. BELLARE, A. BOLDYREVA, AND S. MICALI. Public-key encryption in a multi-user setting: Security proofs and improvements. *Advances in Cryptology—EUROCRYPT 2000* (LNCS 1807) [382], 259–274, 2000.

[36] M. BELLARE, A. DESAI, D. POINTCHEVAL, AND P. ROGAWAY. Relations among notions of security for public-key encryption schemes. *Advances in Cryptology—CRYPTO '98* (LNCS 1462) [270], 26–45, 1998.

[37] M. BELLARE AND P. ROGAWAY. Random oracles are practical: A paradigm for designing efficient protocols. *First ACM Conference on Computer and Communications Security*, 62–73. ACM Press, 1993.

[38] ———. Optimal asymmetric encryption. *Advances in Cryptology—EUROCRYPT '94* (LNCS 950) [110], 92–111, 1995.

[39] ———. The exact security of digital signatures — how to sign with RSA and Rabin. *Advances in Cryptology—EUROCRYPT '96* (LNCS 1070) [306], 399–416, 1996.

[40] ———. Minimizing the use of random oracles in authenticated encryption schemes. *Information and Communications Security '97* (LNCS 1334) [188], 1–16, 1997.

[41] D. BERNSTEIN. Circuits for integer factorization: A proposal. Preprint, 2001. Available from http://cr.yp.to/papers.html.

[42] ———. Floating-point arithmetic and message authentication. Preprint, 2000. Available from http://cr.yp.to/papers.html.

[43] ———. A software implementation of NIST P-224. Presentation at the 5th Workshop on Elliptic Curve Cryptography (ECC 2001), University of Waterloo, October 29-31, 2001. Slides available from http://cr.yp.to/talks.html.

[44] G. BERTONI, J. GUAJARDO, S. KUMAR, G. ORLANDO, C. PAAR, AND T. WOLLINGER. Efficient $GF(p^m)$ arithmetic architectures for cryptographic applications. *Topics in Cryptology—CT-RSA 2003* (LNCS 2612) [230], 158–175, 2003.

[45] T. BETH AND D. GOLLMANN. Algorithm engineering for public key algorithms. *IEEE Journal on Selected Areas in Communications*, 7:458–465, 1989.

[46] I. BIEHL, B. MEYER, AND V. MÜLLER. Differential fault analysis on elliptic curve cryptosystems. *Advances in Cryptology—CRYPTO 2000* (LNCS 1880) [34], 131–146, 2000.

[47] E. BIHAM, editor. *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*. International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Springer-Verlag, 2003.

[48] E. BIHAM AND A. SHAMIR. Differential fault analysis of secret key cryptosystems. *Advances in Cryptology—CRYPTO '97* (LNCS 1294) [235], 513–525, 1997.

[49] I. BLAKE, G. SEROUSSI, AND N. SMART. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[50] S. BLAKE-WILSON AND A. MENEZES. Authenticated Diffie-Hellman key agreement protocols. *Selected Areas in Cryptography—SAC '98* (LNCS 1556) [457], 339–361, 1999.

[51] ———. Unknown key-share attacks on the station-to-station (STS) protocol. *Public Key Cryptography—PKC '99* (LNCS 1560) [206], 154–170, 1999.

[52] D. BLEICHENBACHER. On the security of the KMOV public key cryptosystem. *Advances in Cryptology—CRYPTO '97* (LNCS 1294) [235], 235–248, 1997.

[53] ———. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. *Advances in Cryptology—CRYPTO '98* (LNCS 1462) [270], 1–12, 1998.

[54] D. BONEH. The decision Diffie-Hellman problem. *Algorithmic Number Theory—ANTS-III* (LNCS 1423) [82], 48–63, 1998.

[55] ———, editor. *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*. 23rd Annual International Cryptology Conference, Santa Barbara, California, August 17-21, 2003, Springer-Verlag, 2003.

[56] D. BONEH, R. DEMILLO, AND R. LIPTON. On the importance of checking cryptographic protocols for faults. *Advances in Cryptology—EUROCRYPT '97* (LNCS 1233) [154], 37–51, 1997.

[57] ———. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14:101–119, 2001.

[58] D. BONEH AND M. FRANKLIN. Identity-based encryption from the Weil pairing. *Advances in Cryptology—CRYPTO 2001* (LNCS 2139) [241], 213–229, 2001.

[59] ———. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32:586–615, 2003.

[60] D. BONEH, C. GENTRY, B. LYNN, AND H. SHACHAM. Aggregate and verifiably encrypted signatures from bilinear maps. *Advances in Cryptology—EUROCRYPT 2003* (LNCS 2656) [47], 416–432, 2003.

[61] D. BONEH AND R. LIPTON. Algorithms for black-box fields and their application to cryptography. *Advances in Cryptology—CRYPTO '96* (LNCS 1109) [255], 283–297, 1996.

[62] D. BONEH, B. LYNN, AND H. SHACHAM. Short signatures from the Weil pairing. *Advances in Cryptology—ASIACRYPT 2001* (LNCS 2248) [67], 514–532, 2001.

[63] D. BONEH, I. MIRONOV, AND V. SHOUP. A secure signature scheme from bilinear maps. *Topics in Cryptology—CT-RSA 2003* (LNCS 2612) [230], 98–110, 2003.

[64] D. BONEH AND I. SHPARLINSKI. On the unpredictability of bits of the elliptic curve Diffie-Hellman scheme. *Advances in Cryptology—CRYPTO 2001* (LNCS 2139) [241], 201–212, 2001.

[65] W. BOSMA, editor. *Algorithmic Number Theory—ANTS-IV*, volume 1838 of *Lecture Notes in Computer Science*. 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2000, Springer-Verlag, 2000.

[66] A. BOSSELAERS, R. GOVAERTS, AND J. VANDEWALLE. Comparison of three modular reduction functions. *Advances in Cryptology—CRYPTO '93* (LNCS 773) [453], 175–186, 1994.

[67] C. BOYD, editor. *Advances in Cryptology—ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*. 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Springer-Verlag, 2001.

[68] C. BOYD AND A. MATHURIA. *Protocols for Key Establishment and Authentication*. Springer-Verlag, 2003.

[69] G. BRASSARD, editor. *Advances in Cryptology–CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[70] R. BRENT. An improved Monte Carlo factorization algorithm. *BIT*, 20:176–184, 1980.

[71] E. BRICKELL, editor. *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*. 12th Annual International Cryptology Conference, Santa Barbara, California, August 1992, Springer-Verlag, 1993.

[72] E. BRICKELL, D. GORDON, K. MCCURLEY, AND D. WILSON. Fast exponentiation with precomputation. *Advances in Cryptology—EUROCRYPT '92* (LNCS 658) [398], 200–207, 1993.

[73] E. BRICKELL, D. POINTCHEVAL, S. VAUDENAY, AND M. YUNG. Design validations for discrete logarithm based signature schemes. *Public Key Cryptography—PKC 2000* (LNCS 1751) [207], 276–292, 2000.

[74] É. BRIER AND M. JOYE. Weierstraß elliptic curves and side-channel attacks. *Public Key Cryptography—PKC 2002* (LNCS 2274) [340], 335–345, 2002.

[75] D. BROWN. Generic groups, collision resistance, and ECDSA. Cryptology ePrint Archive: Report 2002/026. Available from http://eprint.iacr.org/2002/026/, February 2002.

[76] D. BROWN AND D. JOHNSON. Formal security proofs for a signature scheme with partial message recovery. *Topics in Cryptology—CT-RSA 2001* (LNCS 2020) [338], 126–142, 2001.

[77] M. BROWN, D. HANKERSON, J. LÓPEZ, AND A. MENEZES. Software implementation of the NIST elliptic curves over prime fields. *Topics in Cryptology—CT-RSA 2001* (LNCS 2020) [338], 250–265, 2001.

[78] D. BRUMLEY AND D. BONEH. Remote timing attacks are practical. *Proceedings of the Twelfth USENIX Security Symposium*. USENIX Association, 2003.

[79] J. BUCHMANN AND H. BAIER. Efficient construction of cryptographically strong elliptic curves. *Progress in Cryptology—INDOCRYPT 2000* (LNCS 1977) [393], 191–202, 2000.

[80] J. BUCHMANN AND H. WILLIAMS. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1:107–118, 1988.

[81] ———. A key exchange system based on real quadratic fields. *Advances in Cryptology–CRYPTO '89* (LNCS 435) [69], 335–343, 1990.

[82] J. BUHLER, editor. *Algorithmic Number Theory—ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*. Third International Symposium, ANTS-III, Portland, Oregon, June 1998, Springer-Verlag, 1998.

[83] R. CANETTI, O. GOLDREICH, AND S. HALEVI. The random oracle methodology, revisited. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 209–218, 1998.

[84] R. CANETTI AND H. KRAWCZYK. Analysis of key-exchange protocols and their use for building secure channels. *Advances in Cryptology—EUROCRYPT 2001* (LNCS 2045) [372], 453–474, 2001.

[85] ———. Security analysis of IKE's signature-based key-exchange protocol. *Advances in Cryptology—CRYPTO 2002* (LNCS 2442) [488], 143–161, 2002.

[86] B. CANVEL, A. HILTGEN, S. VAUDENAY, AND M. VUAGNOUX. Password interception in a SSL/TLS channel. *Advances in Cryptology—CRYPTO 2003* (LNCS 2729) [55], 583–599, 2003.

[87] S. CAVALLAR, B. DODSON, A. LENSTRA, W. LIOEN, P. MONTGOMERY, B. MURPHY, H. TE RIELE, K. AARDAL, J. GILCHRIST, G. GUILLERM, P. LEYLAND, J. MARCHAND, F. MORAIN, A. MUFFETT, C. PUTNAM, C. PUTNAM, AND P. ZIMMERMANN. Factorization of a 512-bit RSA modulus. *Advances in Cryptology—EUROCRYPT 2000* (LNCS 1807) [382], 1–18, 2000.

[88] CERTICOM CORP. ECC Challenge. http://www.certicom.com/resources/ecc_chall/challenge.html, 1997.

[89] A. CHAN AND V. GLIGOR, editors. *Information Security 2002*, volume 2433 of *Lecture Notes in Computer Science*. 5th International Conference, September 30 – October 23, 2002, Sao Paulo, Brazil, Springer-Verlag, 2002.

[90] S. CHANG SHANTZ. From Euclid's GCD to Montgomery multiplication to the great divide. Technical Report SMLI TR-2001-95, Sun Microsystems Laboratories, 2001.

[91] S. CHARI, C. JUTLA, J. RAO, AND P. ROHATGI. Towards sound approaches to counteract power-analysis attacks. *Advances in Cryptology—CRYPTO '99* (LNCS 1666) [480], 398–412, 1999.

[92] L. CHARLAP AND D. ROBBINS. An elementary introduction to elliptic curves. CRD Expository Report 31, Center for Communications Research, Princeton, 1988.

[93] ———. An elementary introduction to elliptic curves II. CRD Expository Report 34, Center for Communications Research, Princeton, 1990.

[94] Y. CHOIE AND J. LEE. Speeding up the scalar multiplication in the jacobians of hyperelliptic curves using frobenius map. *Progress in Cryptology—INDOCRYPT 2002* (LNCS 2551) [317], 285–295, 2002.

[95] B. CHRISTIANSON, B. CRISPO, M. LOMAS, AND M. ROE, editors. *Security Protocols 1997*, volume 1361 of *Lecture Notes in Computer Science*. 5th International Workshop, April 1997, Paris, France, Springer-Verlag, 1998.

[96] D. CHUDNOVSKY AND G. CHUDNOVSKY. Sequences of numbers generated by addition in formal groups and new primality and factoring tests. *Advances in Applied Mathematics*, 7:385–434, 1987.

[97] J. CHUNG, S. SIM, AND P. LEE. Fast implementation of elliptic curve defined over $GF(p^m)$ on CalmRISC with MAC2424 coprocessor. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 57–70, 2000.

[98] M. CIET, T. LANGE, F. SICA, AND J. QUISQUATER. Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms. *Advances in Cryptology—EUROCRYPT 2003* (LNCS 2656) [47], 388–400, 2003.

[99] H. COHEN. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.

[100] H. COHEN, A. MIYAJI, AND T. ONO. Efficient elliptic curve exponentiation using mixed coordinates. *Advances in Cryptology—ASIACRYPT '98* (LNCS 1514) [352], 51–65, 1998.

[101] P. COMBA. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29:526–538, 1990.

[102] D. COPPERSMITH. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30:587–594, 1984.

[103] ———, editor. *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*. 15th Annual International Cryptology Conference, Santa Barbara, California, August 1995, Springer-Verlag, 1995.

[104] J. CORON. Resistance against differential power analysis for elliptic curve cryptosystems. *Cryptographic Hardware and Embedded Systems—CHES '99* (LNCS 1717) [262], 292–302, 1999.

[105] R. CRAMER AND V. SHOUP. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. *Advances in Cryptology—CRYPTO '98* (LNCS 1462) [270], 13–25, 1998.

[106] ———. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, to appear.

[107] I. DAMGÅRD, editor. *Advances in Cryptology—EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*. Workshop on the Theory and Application of Cryptographic Techniques, Aarhus, Denmark, May 1990, Springer-Verlag, 1991.

[108] G. DAVID AND Y. FRANKEL, editors. *Information Security 2001*, volume 2200 of *Lecture Notes in Computer Science*. 4th International Conference, October 1-3, 2001, Malaga, Spain, Springer-Verlag, 2001.

[109] P. DE ROOIJ. Efficient exponentiation using precomputation and vector addition chains. *Advances in Cryptology—EUROCRYPT '94* (LNCS 950) [110], 389–399, 1995.

[110] A. DE SANTIS, editor. *Advances in Cryptology—EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*. Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 1994, Springer-Verlag, 1995.

[111] E. DE WIN, S. MISTER, B. PRENEEL, AND M. WIENER. On the performance of signature schemes based on elliptic curves. *Algorithmic Number Theory—ANTS-III* (LNCS 1423) [82], 252–266, 1998.

[112] B. DEN BOER. Diffie-Hellman is as strong as discrete log for certain primes. *Advances in Cryptology—CRYPTO '88* (LNCS 403) [172], 530–539, 1990.

[113] J. DENEF AND F. VERCAUTEREN. An extension of Kedlaya's algorithm to Artin-Schreier curves in characteristic 2. *Algorithmic Number Theory—ANTS-V* (LNCS 2369) [137], 308–323, 2002.

[114] A. DENT. Adapting the weaknesses of the random oracle model to the generic group model. *Advances in Cryptology—ASIACRYPT 2002* (LNCS 2501) [489], 100–109, 2002.

[115] Y. DESMEDT, editor. *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*. 14th Annual International Cryptology Conference, Santa Barbara, California, August 1994, Springer-Verlag, 1994.

[116] ———, editor. *Public Key Cryptography—PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*. 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, Florida, January 6-8, 2003, Springer-Verlag, 2003.

[117] J. DHEM, F. KOEUNE, P. LEROUX, P. MESTRÉ, J. QUISQUATER, AND J. WILLEMS. A practical implementation of the timing attack. *Smart Card Research and Applications* (LNCS 1820) [387], 175–190, 2000.

[118] C. DIEM. *A Study on Theoretical and Practical Aspects of Weil-Restriction of Varieties*. Ph.D. thesis, University of Essen, Germany, 2001.

[119] ———. The GHS-Attack in odd characteristic. *Journal of the Ramanujan Mathematical Society*, 18:1–32, 2003.

[120] W. DIFFIE. The first ten years of public key cryptology. In Simmons [435], chapter 3, 135–175.

[121] W. DIFFIE AND M. HELLMAN. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[122] W. DIFFIE, P. VAN OORSCHOT, AND M. WIENER. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[123] D. DOLEV, C. DWORK, AND M. NAOR. Non-malleable cryptography. *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, 542–552, 1991.

[124] ———. Non-malleable cryptography. *SIAM Journal on Computing*, 30:391–437, 2000.

[125] J. DOMINGO-FERRER, D. CHAN, AND A. WATSON, editors. *Smart Card Research and Advanced Applications*, volume 180 of *IFIP International Federation for Information Processing*. Fourth Working Conference on Smart Card Research and Advanced Applications (CARDIS 2000), Bristol, UK, September 20-22, 2000, Kluwer, 2000.

[126] A. DURAND. Efficient ways to implement elliptic curve exponentiation on a smart card. *Smart Card Research and Applications* (LNCS 1820) [387], 357–365, 2000.

[127] S. DUSSÉ AND B. KALISKI. A cryptographic library for the Motorola DSP56000. *Advances in Cryptology—EUROCRYPT '90* (LNCS 473) [107], 230–244, 1991.

[128] I. DUURSMA, P. GAUDRY, AND F. MORAIN. Speeding up the discrete log computation on curves with automorphisms. *Advances in Cryptology—ASIACRYPT '99* (LNCS 1716) [274], 103–121, 1999.

[129] K. EISENTRÄGER, K. LAUTER, AND P. MONTGOMERY. Fast elliptic curve arithmetic and improved Weil pairing evaluation. *Topics in Cryptology—CT-RSA 2003* (LNCS 2612) [230], 343–354, 2003.

[130] E. EL MAHASSNI, P. NGUYEN, AND I. SHPARLINSKI. The insecurity of Nyberg-Rueppel and other DSA-like signature schemes with partially known nonces. *Cryptography and Lattices—CaLC 2001*, volume 2146 of *Lecture Notes in Computer Science*, 97–109. Springer-Verlag, 2001.

[131] T. ELGAMAL. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[132] A. ENGE. *Elliptic Curves and Their Applications to Cryptography: An Introduction*. Kluwer Academic Publishers, 1999.

[133] A. ENGE AND P. GAUDRY. A general framework for subexponential discrete logarithm algorithms. *Acta Arithmetica*, 102:83–103, 2002.

[134] M. ERNST, M. JUNG, F. MADLENER, S. HUSS, AND R. BLÜMEL. A reconfigurable system on chip implementation for elliptic curve cryptography over $GF(2^n)$. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 381–399, 2002.

[135] J. FEIGENBAUM, editor. *Advances in Cryptology—CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[136] N. FERGUSON AND B. SCHNEIER. *Practical Cryptography*. Wiley, 2003.

[137] C. FIEKER AND D. KOHEL, editors. *Algorithmic Number Theory—ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*. 5th International Symposium, ANTS-V, Sydney, Australia, July 2002, Springer-Verlag, 2002.

[138] FIPS 180-2. Secure Hash Standard. Federal Information Processing Standards Publication 180-2, National Institute of Standards and Technology, 2002.

[139] FIPS 186. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186, National Institute of Standards and Technology, 1994.

[140] FIPS 186-2. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology, 2000.

[141] FIPS 197. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, National Institute of Standards and Technology, 2001.

[142] FIPS 198. HMAC – Keyed-Hash Message Authentication. Federal Information Processing Standards Publication 198, National Institute of Standards and Technology, 2002.

[143] P. FLAJOLET AND A. ODLYZKO. Random mapping statistics. *Advances in Cryptology —EUROCRYPT '89* (LNCS 434) [388], 329–354, 1990.

[144] K. FONG, D. HANKERSON, J. LÓPEZ, AND A. MENEZES. Field inversion and point halving revisited. Technical Report CORR 2003-18, Department of Combinatorics and Optimization, University of Waterloo, Canada, 2003.

[145] W. FORD AND M. BAUM. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Prentice Hall, 2nd edition, 2000.

[146] M. FOUQUET, P. GAUDRY, AND R. HARLEY. An extension of Satoh's algorithm and its implementation. *Journal of the Ramanujan Mathematical Society*, 15:281–318, 2000.

[147] ———. Finding secure curves with the Satoh-FGH algorithm and an early-abort strategy. *Advances in Cryptology—EUROCRYPT 2001* (LNCS 2045) [372], 14–29, 2001.

[148] Y. FRANKEL, editor. *Financial Cryptography—FC 2000*, volume 1962 of *Lecture Notes in Computer Science*. 4th International Conference, FC 2000, Anguilla, British West Indies, February 2000, Springer-Verlag, 2001.

[149] G. FREY. Applications of arithmetical geometry to cryptographic constructions. *Proceedings of the Fifth International Conference on Finite Fields and Applications*, 128–161. Springer-Verlag, 2001.

[150] G. FREY AND H. RÜCK. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62:865–874, 1994.

[151] H. FRIUM. The group law on elliptic curves on Hesse form. *Finite Fields with Applications to Coding Theory, Cryptography and Related Areas*, 123–151. Springer-Verlag, 2002.

[152] E. FUJISAKI AND T. OKAMOTO. Secure integration of asymmetric and symmetric encryption schemes. *Advances in Cryptology—CRYPTO '99* (LNCS 1666) [480], 537–554, 1999.

[153] E. FUJISAKI, T. OKAMOTO, D. POINTCHEVAL, AND J. STERN. RSA-OAEP is secure under the RSA assumption. *Advances in Cryptology—CRYPTO 2001* (LNCS 2139) [241], 260–274, 2001.

[154] W. FUMY, editor. *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 1997, Springer-Verlag, 1997.

[155] S. GALBRAITH. Supersingular curves in cryptography. *Advances in Cryptology—ASIACRYPT 2001* (LNCS 2248) [67], 495–513, 2001.

[156] S. GALBRAITH, F. HESS, AND N. SMART. Extending the GHS Weil descent attack. *Advances in Cryptology—EUROCRYPT 2002* (LNCS 2332) [248], 29–44, 2002.

[157] S. GALBRAITH, S. PAULUS, AND N. SMART. Arithmetic on superelliptic curves. *Mathematics of Computation*, 71:393–405, 2002.

[158] S. GALBRAITH AND N. SMART. A cryptographic application of Weil descent. *Cryptography and Coding*, volume 1746 of *Lecture Notes in Computer Science*, 191–200. Springer-Verlag, 1999.

[159] R. GALLANT, R. LAMBERT, AND S. VANSTONE. Improving the parallelized Pollard lambda search on anomalous binary curves. *Mathematics of Computation*, 69:1699–1705, 2000.

[160] ———. Faster point multiplication on elliptic curves with efficient endomorphisms. *Advances in Cryptology—CRYPTO 2001* (LNCS 2139) [241], 190–200, 2001.

[161] K. GANDOLFI, C. MOURTEL, AND F. OLIVIER. Electromagnetic analysis: concrete results. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 251–261, 2001.

[162] L. GAO, S. SHRIVASTAVA, AND G. SOBELMAN. Elliptic curve scalar multiplier design using FPGAs. *Cryptographic Hardware and Embedded Systems—CHES '99* (LNCS 1717) [262], 257–268, 1999.

[163] P. GAUDRY. An algorithm for solving the discrete log problem in hyperelliptic curves. *Advances in Cryptology—EUROCRYPT 2000* (LNCS 1807) [382], 19–34, 2000.

[164] ———. A comparison and a combination of SST and AGM algorithms for counting points of elliptic curves in characteristic 2. *Advances in Cryptology—ASIACRYPT 2002* (LNCS 2501) [489], 311–327, 2002.

[165] P. GAUDRY AND N. GÜREL. An extension of Kedlaya's point-counting algorithm to superelliptic curves. *Advances in Cryptology—ASIACRYPT 2001* (LNCS 2248) [67], 480–494, 2001.

[166] P. GAUDRY AND R. HARLEY. Counting points on hyperelliptic curves over finite fields. *Algorithmic Number Theory—ANTS-IV* (LNCS 1838) [65], 313–332, 2000.

[167] P. GAUDRY, F. HESS, AND N. SMART. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15:19–46, 2002.

[168] C. GEBOTYS AND R. GEBOTYS. Secure elliptic curve implementations: An analysis of resistance to power-attacks in a DSP. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 114–128, 2002.

[169] W. GEISELMANN AND R. STEINWANDT. A dedicated sieving hardware. *Public Key Cryptography—PKC 2003* (LNCS 2567) [116], 267–278, 2003.

[170] C. GENTRY AND A. SILVERBERG. Hierarchical ID-based cryptography. *Advances in Cryptology—ASIACRYPT 2002* (LNCS 2501) [489], 548–566, 2002.

[171] R. GERBER. *The Software Optimization Cookbook – High-Performance Recipes for the Intel Architecture*. Intel Press, 2002.

[172] S. GOLDWASSER, editor. *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[173] S. GOLDWASSER AND J. KILIAN. Almost all primes can be quickly certified. *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, 316–329, 1986.

[174] S. GOLDWASSER AND S. MICALI. Probabilistic encryption. *Journal of Computer and System Sciences*, 29:270–299, 1984.

[175] S. GOLDWASSER, S. MICALI, AND R. RIVEST. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, 1988.

[176] G. GONG AND L. HARN. Public-key cryptosystems based on cubic finite field extensions. *IEEE Transactions on Information Theory*, 45:2601–2605, 1999.

[177] J. GOODMAN AND A. CHANDRAKASAN. An energy efficient reconfigurable public-key cryptography processor architecture. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 175–190, 2000.

[178] D. GORDON. Discrete logarithms in $GF(p)$ using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6:124–138, 1993.

[179] ———. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.

[180] L. GOUBIN. A refined power-analysis attack on elliptic curve cryptosystems. *Public Key Cryptography—PKC 2003* (LNCS 2567) [116], 199–210, 2003.

[181] J. GROSSSCHÄDL. A bit-serial unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 202–219, 2001.

[182] J. GUAJARDO, R. BLÜMEL, U. KRIEGER, AND C. PAAR. Efficient implementation of elliptic curve cryptosystems on the TI MSP430x33x family of microcontrollers. *Public Key Cryptography—PKC 2001* (LNCS 1992) [244], 365–382, 2001.

[183] J. GUAJARDO AND C. PAAR. Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. *Designs, Codes and Cryptography*, 25:207–216, 2002.

[184] L. GUILLOU AND J. QUISQUATER, editors. *Advances in Cryptology—EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995, Springer-Verlag, 1995.

[185] C. GÜNTHER, T. LANGE, AND A. STEIN. Speeding up the arithmetic on Koblitz curves of genus two. *Selected Areas in Cryptography—SAC 2000* (LNCS 2012) [455], 106–117, 2001.

[186] N. GURA, S. CHANG SHANTZ, H. EBERLE, D. FINCHELSTEIN, S. GUPTA, V. GUPTA, AND D. STEBILA. An end-to-end systems approach to elliptic curve cryptography. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 349–365, 2002.

[187] A. GUTUB, A. TENCA, E. SAVAŞ, AND C. KOÇ. Scalable and unified hardware to compute Montgomery inverse in $GF(p)$ and $GF(2^n)$. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 484–499, 2002.

[188] Y. HAN, T. OKAMOTO, AND S. QING, editors. *Information and Communications Security '97*, volume 1334 of *Lecture Notes in Computer Science*. First Inernational Conference, November 11-14, 1997, Beijing, China, Springer-Verlag, 1997.

[189] D. HANKERSON, J. LÓPEZ, AND A. MENEZES. Software implementation of elliptic curve cryptography over binary fields. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 1–24, 2000.

[190] D. HARKINS AND D. CARREL. The Internet Key Exchange (IKE). Internet Request for Comments 2409, Available from http://www.ietf.org/rfc/rfc2409.txt, November 1998.

[191] R. HARLEY. The Elliptic Curve Discrete Logarithms Project. http://pauillac.inria.fr/~harley/ecdl/, 1997.

[192] ———. Asymptotically optimal p-adic point-counting. Contribution to the Number Theory List, 2002.

[193] M. HASAN. Power analysis attacks and algorithmic approaches to their countermeasures for Koblitz curve cryptosystems. *IEEE Transactions on Computers*, 50:1071–1083, 2001.

[194] T. HASEGAWA, J. NAKAJIMA, AND M. MATSUI. A practical implementation of elliptic curve cryptosystems over $GF(p)$ on a 16-bit microcomputer. *Public Key Cryptography—PKC '98* (LNCS 1431) [205], 182–194, 1998.

[195] J. HÅSTAD. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17:336–341, 1988.

[196] F. HESS. The GHS attack revisited. *Advances in Cryptology—EUROCRYPT 2003* (LNCS 2656) [47], 374–387, 2003.

[197] A. HEVIA AND M. KIWI. Strength of two Data Encryption Standard implementations under timing attacks. *ACM Transactions on Information and System Security*, 2:416–437, 1999.

[198] Y. HITCHCOCK AND P. MONTAGUE. A new elliptic curve scalar multiplication algorithm to resist simple power analysis. *Information Security and Privacy 2002* (LNCS 2384) [30], 214–225, 2002.

[199] J. HORWITZ AND B. LYNN. Toward hierarchical identity-based encryption. *Advances in Cryptology—EUROCRYPT 2002* (LNCS 2332) [248], 466–481, 2002.

[200] R. HOUSLEY AND T. POLK. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. Wiley, 2001.

[201] E. HOWE. On the group orders of elliptic curves over finite fields. *Compositio Mathematica*, 85:229–247, 1993.

[202] N. HOWGRAVE-GRAHAM AND N. SMART. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23:283–290, 2001.

[203] IEEE P1363A. Standard Specifications for Public-Key Cryptography—Amendment 1: Additional Techniques, working draft 12bis, May 12 2003.

[204] IEEE STD 1363-2000. IEEE Standard Specifications for Public-Key Cryptography, 2000.

[205] H. IMAI AND Y. ZHENG, editors. *Public Key Cryptography—PKC '98*, volume 1431 of *Lecture Notes in Computer Science*. First International Workshop on Practice and Theory in Public Key Cryptography, Pacifico Yokohama, Japan, February 1998, Springer-Verlag, 1998.

[206] ——, editors. *Public Key Cryptography—PKC '99*, volume 1560 of *Lecture Notes in Computer Science*. Second International Workshop on Practice and Theory in Public Key Cryptography, Kamakura, Japan, March 1999, Springer-Verlag, 1999.

[207] ——, editors. *Public Key Cryptography—PKC 2000*, volume 1751 of *Lecture Notes in Computer Science*. Third International Workshop on Practice and Theory in Public Key Cryptosystems, Melbourne, Australia, January 2000, Springer-Verlag, 2000.

[208] INTEL CORPORATION. *The Complete Guide to MMX Technology*. McGraw-Hill, 1997. Contributing authors: D. Bistry, C. Delong, M. Gutman, M. Julier, M. Keith, L. Mennemeier, M. Mittal, A. Peleg, and U. Weiser.

[209] ——. *Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual*, 2001. Number 248966-04, available from http://developer.intel.com.

[210] ——. *IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, 2002. Number 245470-007, available from http://developer.intel.com.

[211] ISO/IEC 15946-1. Information Technology – Security Techniques – Cryptographic Techniques Based on Elliptic Curves – Part 1: General, 2002.

[212] ISO/IEC 15946-2. Information Technology – Security Techniques – Cryptographic Techniques Based on Elliptic Curves – Part 2: Digital Signatures, 2002.

[213] ISO/IEC 15946-3. Information Technology – Security Techniques – Cryptographic Techniques Based on Elliptic Curves – Part 3: Key Establishment, 2002.

[214] ISO/IEC 15946-4. Information Technology – Security Techniques – Cryptographic Techniques Based on Elliptic Curves – Part 4: Digital Signatures Giving Message Recovery, draft 2003.

[215] ISO/IEC 18033-2. Information Technology – Security Techniques – Encryption Algorithms – Part 2: Asymmetric Ciphers, draft 2002.

[216] K. ITOH, M. TAKENAKA, N. TORII, S. TEMMA, AND Y. KURIHARA. Fast implementation of public-key cryptography on a DSP TMS320C6201. *Cryptographic Hardware and Embedded Systems—CHES '99* (LNCS 1717) [262], 61–72, 1999.

[217] T. ITOH AND S. TSUJII. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation*, 78:171–177, 1988.

[218] T. IZU, J. KOGURE, M. NORO, AND K. YOKOYAMA. Efficient implementation of Schoof's algorithm. *Advances in Cryptology—ASIACRYPT '98* (LNCS 1514) [352], 66–79, 1998.

[219] T. IZU, B. MÖLLER, AND T. TAKAGI. Improved elliptic curve multiplication methods resistant against side channel attacks. *Progress in Cryptology—INDOCRYPT 2002* (LNCS 2551) [317], 296–313, 2002.

[220] T. IZU AND T. TAKAGI. A fast parallel elliptic curve multiplication resistant against side channel attacks. *Public Key Cryptography—PKC 2002* (LNCS 2274) [340], 280–296, 2002.

[221] ———. Exceptional procedure attack on elliptic curve cryptosystems. *Public Key Cryptography—PKC 2003* (LNCS 2567) [116], 224–239, 2003.

[222] M. JACOBSON, N. KOBLITZ, J. SILVERMAN, A. STEIN, AND E. TESKE. Analysis of the xedni calculus attack. *Designs, Codes and Cryptography*, 20:41–64, 2000.

[223] M. JACOBSON, A. MENEZES, AND A. STEIN. Solving elliptic curve discrete logarithm problems using Weil descent. *Journal of the Ramanujan Mathematical Society*, 16:231–260, 2001.

[224] D. JOHNSON. Key validation. Research contribution to IEEE P1363. Available from http://grouper.ieee.org/groups/1363/Research, 1997.

[225] ———. Public key validation: A piece of the PKI puzzle. Research contribution to IEEE P1363. Available from http://grouper.ieee.org/groups/1363/Research, 2000.

[226] D. JOHNSON, A. MENEZES, AND S. VANSTONE. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1:36–63, 2001.

[227] A. JOUX. A one round protocol for tripartite Diffie-Hellman. *Algorithmic Number Theory—ANTS-IV* (LNCS 1838) [65], 385–393, 2000.

[228] A. JOUX AND R. LERCIER. Improvements to the general Number Field Sieve for discrete logarithms in prime fields: A comparison with the Gaussian integer method. *Mathematics of Computation*, 72:953–967, 2003.

[229] A. JOUX AND K. NGUYEN. Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups. *Journal of Cryptology*, to appear.

[230] M. JOYE, editor. *Topics in Cryptology—CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*. The Cryptographer's Track at the RSA Conference 2003, San Francisco, California, April 13-17, 2003, Springer-Verlag, 2003.

[231] M. JOYE AND J. QUISQUATER. Hessian elliptic curves and side-channel attacks. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 402–410, 2001.

[232] M. JOYE AND C. TYMEN. Protections against differential analysis for elliptic curve cryptography – an algebraic approach. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 377–390, 2001.

[233] B. KALISKI. One-way permutations on elliptic curves. *Journal of Cryptology*, 3:187–199, 1991.

[234] ———. The Montgomery inverse and its applications. *IEEE Transactions on Computers*, 44:1064–1065, 1995.

[235] ———, editor. *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*. 17th Annual International Cryptology Conference, Santa Barbara, California, August 1997, Springer-Verlag, 1997.

[236] ———. A chosen message attack on Demytko's elliptic curve cryptosystem. *Journal of Cryptology*, 10:71–72, 1997.

[237] ———. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and System Security*, 4:275–288, 2001.

[238] B. KALISKI, Ç. KOÇ, AND C. PAAR, editors. *Cryptographic Hardware and Embedded Systems—CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[239] A. KARATSUBA AND Y. OFMAN. Multiplication of multidigit numbers on automata. *Soviet Physics — Doklady*, 7:595–596, 1963.

[240] K. KEDLAYA. Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology. *Journal of the Ramanujan Mathematical Society*, 16:323–338, 2001.

[241] J. KILIAN, editor. *Advances in Cryptology—CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*. 21st Annual International Cryptology Conference, Santa Barbara, California, August 19-23, 2001, Springer-Verlag, 2001.

[242] D. KIM AND S. LIM. Integer decomposition for fast scalar multiplication on elliptic curves. *Selected Areas in Cryptography—SAC 2002* (LNCS 2595) [349], 13–20, 2003.

[243] H. KIM, J. PARK, J. CHEON, J. PARK, J. KIM, AND S. HAHN. Fast elliptic curve point counting using Gaussian normal basis. *Algorithmic Number Theory—ANTS-V* (LNCS 2369) [137], 292–207, 2002.

[244] K. KIM, editor. *Public Key Cryptography—PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*. 4th International Workshop on Practice and Theory in Public Key Cryptosystems, Cheju Island, Korea, February 13-15, 2001, Springer-Verlag, 2001.

[245] K. KIM AND T. MATSUMOTO, editors. *Advances in Cryptology—ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptology and Information Security, Kyongju, Korea, November 1996, Springer-Verlag, 1996.

[246] B. KING. An improved implementation of elliptic curves over $GF(2^n)$ when using projective point arithmetic. *Selected Areas in Cryptography—SAC 2001* (LNCS 2259) [468], 134–150, 2001.

[247] E. KNUDSEN. Elliptic scalar multiplication using point halving. *Advances in Cryptology—ASIACRYPT '99* (LNCS 1716) [274], 135–149, 1999.

[248] L. KNUDSEN, editor. *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*. International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 – May 2, 2002, Springer-Verlag, 2002.

[249] D. KNUTH. *The Art of Computer Programming—Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1998.

[250] N. KOBLITZ. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[251] ———. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1:139–150, 1989.

[252] ———. Constructing elliptic curve cryptosystems in characteristic 2. *Advances in Cryptology—CRYPTO '90* (LNCS 537) [320], 156–167, 1991.

[253] ———. CM-curves with good cryptographic properties. *Advances in Cryptology—CRYPTO '91* (LNCS 576) [135], 279–287, 1992.

[254] ———. *A Course in Number Theory and Cryptography*. Springer-Verlag, 2nd edition, 1994.

[255] ———, editor. *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*. 16th Annual International Cryptology Conference, Santa Barbara, California, August 1996, Springer-Verlag, 1996.

[256] ———. An elliptic curve implementation of the finite field digital signature algorithm. *Advances in Cryptology—CRYPTO '98* (LNCS 1462) [270], 327–337, 1998.

[257] ———. Good and bad uses of elliptic curves in cryptography. *Moscow Mathematical Journal*, 2:693–715, 2002.

[258] Ç. KOÇ. High-speed RSA implementation. Technical Report TR-201, RSA Laboratories, 1994.

[259] Ç. KOÇ AND T. ACAR. Montgomery multiplication in $GF(2^k)$. *Designs, Codes and Cryptography*, 14:57–69, 1998.

[260] Ç. KOÇ, T. ACAR, AND B. KALISKI. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16:26–33, 1996.

[261] Ç. KOÇ, D. NACCACHE, AND C. PAAR, editors. *Cryptographic Hardware and Embedded Systems—CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[262] Ç. KOÇ AND C. PAAR, editors. *Cryptographic Hardware and Embedded Systems—CHES '99*, volume 1717 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[263] ———, editors. *Cryptographic Hardware and Embedded Systems—CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[264] P. KOCHER. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Advances in Cryptology—CRYPTO '96* (LNCS 1109) [255], 104–113, 1996.

[265] P. KOCHER, J. JAFFE, AND B. JUN. Differential power analysis. *Advances in Cryptology—CRYPTO '99* (LNCS 1666) [480], 388–397, 1999.

[266] I. KOREN. *Computer Arithmetic Algorithms*. A.K. Peters, 2nd edition, 2002.

[267] K. KOYAMA, U. MAURER, T. OKAMOTO, AND S. VANSTONE. New public-key schemes based on elliptic curves over the ring $\mathbb{Z}_n$. *Advances in Cryptology—CRYPTO '91* (LNCS 576) [135], 252–266, 1992.

[268] D. KRAVITZ. Digital signature algorithm. U.S. patent # 5,231,668, 1993.

[269] H. KRAWCZYK. SKEME: A versatile secure key exchange mechanism for internet. *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, 114–127, 1996.

[270] ———, editor. *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*. 18th Annual International Cryptology Conference, Santa Barbara, California, August 1998, Springer-Verlag, 1998.

[271] F. KUHN AND R. STRUIK. Random walks revisited: Extensions of Pollard's rho algorithm for computing multiple discrete logarithms. *Selected Areas in Cryptography—SAC 2001* (LNCS 2259) [468], 212–229, 2001.

[272] M. KUHN AND R. ANDERSON. Soft tempest: Hidden data transmission using electromagnetic emanations. *Information Hiding—IH '98* (LNCS 1525) [22], 124–142, 1998.

[273] K. KUROSAWA, K. OKAKA, AND S. TSUJII. Low exponent attack against elliptic curve RSA. *Advances in Cryptology—ASIACRYPT '94* (LNCS 917) [373], 376–383, 1995.

[274] K. LAM AND E. OKAMOTO, editors. *Advances in Cryptology—ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptology and Information Security, Singapore, November 1999, Springer-Verlag, 1999.

[275] L. LAW, A. MENEZES, M. QU, J. SOLINAS, AND S. VANSTONE. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.

[276] G. LAY AND H. ZIMMER. Constructing elliptic curves with given group order over large finite fields. *Algorithmic Number Theory—ANTS-I* (LNCS 877) [3], 250–263, 1994.

[277] P. LEE AND C. LIM, editors. *Information Security and Cryptology 2002*, volume 2587 of *Lecture Notes in Computer Science*. 5th International Conference, November 28-29, 2002, Seoul, Korea, Springer-Verlag, 2003.

[278] D. LEHMER. Euclid's algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.

[279] A. LENSTRA. Unbelievable security—matching AES security using public key systems. *Advances in Cryptology—ASIACRYPT 2001* (LNCS 2248) [67], 67–86, 2001.

[280] A. LENSTRA AND H. LENSTRA, editors. *The Development of the Number Field Sieve*. Springer-Verlag, 1993.

[281] A. LENSTRA AND A. SHAMIR. Analysis and optimization of the TWINKLE factoring device. *Advances in Cryptology—EUROCRYPT 2000* (LNCS 1807) [382], 35–52, 2000.

[282] A. LENSTRA, A. SHAMIR, J. TOMLINSON, AND E. TROMER. Analysis of Bernstein's factorization circuit. *Advances in Cryptology—ASIACRYPT 2002* (LNCS 2501) [489], 1–26, 2002.

[283] A. LENSTRA AND E. VERHEUL. The XTR public key system. *Advances in Cryptology—CRYPTO 2000* (LNCS 1880) [34], 1–19, 2000.

[284] ———. Selecting cryptographic key sizes. *Journal of Cryptology*, 14:255–293, 2001.

[285] H. LENSTRA. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[286] P. LEONG AND K. LEUNG. A microcoded elliptic curve processor using FPGA technology. *IEEE Transactions on VLSI Systems*, 10:550–559, 2002.

[287] R. LERCIER. Finding good random elliptic curves for cryptosystems defined over $\mathbb{F}_{2^n}$. *Advances in Cryptology—EUROCRYPT '97* (LNCS 1233) [154], 379–392, 1997.

[288] R. LERCIER AND D. LUBICZ. Counting points on elliptic curves over finite fields of small characteristic in quasi quadratic time. *Advances in Cryptology—EUROCRYPT 2003* (LNCS 2656) [47], 360–373, 2003.

[289] R. LERCIER AND F. MORAIN. Counting the number of points on elliptic curves over finite fields: strategies and performances. *Advances in Cryptology—EUROCRYPT '95* (LNCS 921) [184], 79–94, 1995.

[290] S. LEVY. *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*. Penguin Books, 2001.

[291] P. LIARDET AND N. SMART. Preventing SPA/DPA in ECC systems using the Jacobi form. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 391–401, 2001.

[292] R. LIDL AND H. NIEDERREITER. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, revised edition, 1994.

[293] C. LIM AND H. HWANG. Fast implementation of elliptic curve arithmetic in $GF(p^n)$. *Public Key Cryptography—PKC 2000* (LNCS 1751) [207], 405–421, 2000.

[294] ———. Speeding up elliptic scalar multiplication with precomputation. *Information Security and Cryptology '99* (LNCS 1787) [448], 102–119, 2000.

[295] C. LIM AND P. LEE. More flexible exponentiation with precomputation. *Advances in Cryptology—CRYPTO '94* (LNCS 839) [115], 95–107, 1994.

[296] ———. A key recovery attack on discrete log-based schemes using a prime order subgroup. *Advances in Cryptology—CRYPTO '97* (LNCS 1294) [235], 249–263, 1997.

[297] ———. A study on the proposed Korean digital signature algorithm. *Advances in Cryptology—ASIACRYPT '98* (LNCS 1514) [352], 175–186, 1998.

[298] H. LIPMAA. IDEA: A cipher for multimedia architectures? *Selected Areas in Cryptography—SAC '98* (LNCS 1556) [457], 248–263, 1999.

[299] J. LÓPEZ AND R. DAHAB. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. *Cryptographic Hardware and Embedded Systems—CHES '99* (LNCS 1717) [262], 316–327, 1999.

[300] ———. Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. *Selected Areas in Cryptography—SAC '98* (LNCS 1556) [457], 201–212, 1999.

[301] ———. High-speed software multiplication in $\mathbb{F}_{2^m}$. *Progress in Cryptology—INDOCRYPT 2000* (LNCS 1977) [393], 203–212, 2000.

[302] J. LOUGHRY AND D. UMPHRESS. Information leakage from optical emanations. *ACM Transactions on Information and System Security*, 5:262–289, 2002.

[303] J. MANGER. A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS #1 v2.0. *Advances in Cryptology—CRYPTO 2001* (LNCS 2139) [241], 230–238, 2001.

[304] M. MAURER, A. MENEZES, AND E. TESKE. Analysis of the GHS Weil descent attack on the ECDLP over characteristic two finite fields of composite degree. *LMS Journal of Computation and Mathematics*, 5:127–174, 2002.

[305] U. MAURER. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. *Advances in Cryptology—CRYPTO '94* (LNCS 839) [115], 271–281, 1994.

[306] ———, editor. *Advances in Cryptology—EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 1996, Springer-Verlag, 1996.

[307] U. MAURER AND S. WOLF. The Diffie-Hellman protocol. *Designs, Codes and Cryptography*, 19:147–171, 2000.

[308] D. MAY, H. MULLER, AND N. SMART. Non-deterministic processors. *Information Security and Privacy 2001* (LNCS 2119) [465], 115–129, 2001.

[309] ———. Random register renaming to foil DPA. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 28–38, 2001.

[310] K. MCCURLEY. A key distribution system equivalent to factoring. *Journal of Cryptology*, 1:95–105, 1988.

[311] R. MCELIECE. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987.

[312] W. MEIER AND O. STAFFELBACH. Efficient multiplication on certain nonsupersingular elliptic curves. *Advances in Cryptology—CRYPTO '92* (LNCS 740) [71], 333–344, 1993.

[313] A. MENEZES. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[314] A. MENEZES, T. OKAMOTO, AND S. VANSTONE. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.

[315] A. MENEZES AND M. QU. Analysis of the Weil descent attack of Gaudry, Hess and Smart. *Topics in Cryptology—CT-RSA 2001* (LNCS 2020) [338], 308–318, 2001.

[316] A. MENEZES, M. QU, AND S. VANSTONE. Key agreement and the need for authentication. Presentation at PKS '95, Toronto, Canada, November 1995.

[317] A. MENEZES AND P. SARKAR, editors. *Progress in Cryptology—INDOCRYPT 2002*, volume 2551 of *Lecture Notes in Computer Science*. Third International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002, Springer-Verlag, 2002.

[318] A. MENEZES, E. TESKE, AND A. WENG. Weak fields for ECC. Cryptology ePrint Archive: Report 2003/128. Available from http://eprint.iacr.org/2003/128/, June 2003.

[319] A. MENEZES, P. VAN OORSCHOT, AND S. VANSTONE. *Handbook of Applied Cryptography*. CRC Press, 1996.

[320] A. MENEZES AND S. VANSTONE, editors. *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[321] R. MERKLE. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, 1978.

[322] R. MERKLE AND M. HELLMAN. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24:525–530, 1978.

[323] T. MESSERGES, E. DABBISH, AND R. SLOAN. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, 51:541–552, 2002.

[324] J. MESTRE. Formules explicites et minoration de conducteurs de variétés algébriques. *Compositio Mathematica*, 58:209–232, 1986.

[325] V. MILLER. Use of elliptic curves in cryptography. *Advances in Cryptology—CRYPTO '85* (LNCS 218) [483], 417–426, 1986.

[326] B. MÖLLER. Algorithms for multi-exponentiation. *Selected Areas in Cryptography—SAC 2001* (LNCS 2259) [468], 165–180, 2001.

[327] ———. Securing elliptic curve point multiplication against side-channel attacks. *Information Security 2001* (LNCS 2200) [108], 324–334, 2001.

[328] ———. Parallelizable elliptic curve point multiplication method with resistance against side-channel attacks. *Information Security 2002* (LNCS 2433) [89], 402–413, 2002.

[329] ———. Improved techniques for fast exponentiation. *Information Security and Cryptology 2002* (LNCS 2587) [277], 298–312, 2003.

[330] P. MONTGOMERY. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.

[331] ———. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.

[332] S. MOORE. Using streaming SIMD extensions (SSE2) to perform big multiplications. Application Note AP-941, Intel Corporation, 2000. Version 2.0, Order Number 248606-001.

[333] F. MORAIN AND J. OLIVOS. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Informatique Théorique et Applications*, 24:531–544, 1990.

[334] V. MÜLLER. Fast multiplication on elliptic curves over small fields of characteristic two. *Journal of Cryptology*, 11:219–234, 1998.

[335] ——. Efficient point multiplication for elliptic curves over special optimal extension fields. *Public-Key Cryptography and Computational Number Theory*, 197–207. de Gruyter, 2001.

[336] V. MÜLLER, S. VANSTONE, AND R. ZUCCHERATO. Discrete logarithm based cryptosystems in quadratic function fields of characteristic 2. *Designs, Codes and Cryptography*, 14:159–178, 1998.

[337] R. MULLIN, I. ONYSZCHUK, S. VANSTONE, AND R. WILSON. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22:149–161, 1988/89.

[338] D. NACCACHE, editor. *Topics in Cryptology—CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*. The Cryptographers' Track at RSA Conference 2001, San Francisco, California, April 8-12, 2001, Springer-Verlag, 2001.

[339] D. NACCACHE AND D. M'RAÏHI. Cryptographic smart cards. *IEEE Micro*, 16(3):14–24, June 1996.

[340] D. NACCACHE AND P. PAILLIER, editors. *Public Key Cryptography—PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*. 5th International Workshop on Practice and Theory in Public Key Cryptosystems, Paris, France, February 2002, Springer-Verlag, 2002.

[341] D. NACCACHE AND J. STERN. Signing on a postcard. *Financial Cryptography—FC 2000* (LNCS 1962) [148], 121–135, 2001.

[342] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. NIST Special Publication 800-56: Recommendation on key establishment schemes, Draft 2.0. Available from http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html, January 2003.

[343] NATIONAL SECURITY AGENCY. NACSIM 5000 Tempest Fundamentals (U). Fort George G. Meade, Maryland, USA. Available from http://cryptome.org/nacsim-5000.htm.

[344] V. NECHAEV. Complexity of a determinate algorithm for the discrete logarithm problem. *Mathematical Notes*, 55:165–172, 1994.

[345] P. NGUYEN AND I. SHPARLINSKI. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15:151–176, 2002.

[346] ——. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, to appear.

[347] M. NIELSEN AND I. CHUANG. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[348] P. NING AND Y. YIN. Efficient software implementation for finite field multiplication in normal basis. *Information and Communications Security 2001* (LNCS 2229) [385], 177–189, 2001.

[349] K. NYBERG AND H. HEYS, editors. *Selected Areas in Cryptography—SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*. 9th Annual International Workshop, St. John's, Newfoundland, Canada, August 15-16, 2002, Springer-Verlag, 2003.

[350] K. NYBERG AND R. RUEPPEL. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes and Cryptography*, 7:61–81, 1996.

[351] A. ODLYZKO, editor. *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987.

[352] K. OHTA AND D. PEI, editors. *Advances in Cryptology—ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, October 1998, Springer-Verlag, 1998.

[353] S. OKADA, N. TORII, K. ITOH, AND M. TAKENAKA. Implementation of elliptic curve cryptographic coprocessor over $GF(2^m)$ on an FPGA. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 25–40, 2000.

[354] T. OKAMOTO. Provably secure and practical identification schemes and corresponding signature schemes. *Advances in Cryptology—CRYPTO '92* (LNCS 740) [71], 31–53, 1993.

[355] ———, editor. *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*. 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 2000, Springer-Verlag, 2000.

[356] T. OKAMOTO AND D. POINTCHEVAL. The gap-problems: A new class of problems for the security of cryptographic schemes. *Public Key Cryptography—PKC 2001* (LNCS 1992) [244], 104–118, 2001.

[357] ———. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. *Topics in Cryptology—CT-RSA 2001* (LNCS 2020) [338], 159–175, 2001.

[358] K. OKEYA AND K. SAKURAI. Power analysis breaks elliptic curve cryptosystems even secure against the timing attack. *Progress in Cryptology—INDOCRYPT 2000* (LNCS 1977) [393], 178–190, 2000.

[359] ———. Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 126–141, 2001.

[360] OPEN MOBILE ALLIANCE LTD. Wireless Transport Layer Security. Version 06-Apr-2001.

[361] G. ORLANDO AND C. PAAR. A high-performance reconfigurable elliptic curve processor for $GF(2^m)$. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 41–56, 2000.

[362] ———. A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 348–363, 2001.

[363] H. ORMAN. The OAKLEY key determination protocol. Internet Request for Comments 2412, Available from http://www.ietf.org/rfc/rfc2412.txt, November 1998.

[364] E. OSWALD. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 82–97, 2002.

[365] C. PAAR AND P. SORIA-RODRIGUEZ. Fast arithmetic architectures for public-key algorithms over Galois fields $GF((2^n)^m)$. *Advances in Cryptology—EUROCRYPT '97* (LNCS 1233) [154], 363–378, 1997.

[366] D. PAGE AND N. SMART. Hardware implementation of finite fields of characteristic three. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 529–539, 2002.

[367] C. PANDU RANGAN AND C. DING, editors. *Progress in Cryptology—INDOCRYPT 2001*, volume 2247 of *Lecture Notes in Computer Science*. Second International Conference on Cryptology in India, Chennai, India, December 16-20, 2001, Springer-Verlag, 2001.

[368] Y. PARK, S. JEONG, C. KIM, AND J. LIM. An alternate decomposition of an integer for faster point multiplication on certain elliptic curves. *Public Key Cryptography—PKC 2002* (LNCS 2274) [340], 323–334, 2002.

[369] Y. PARK, S. JEONG, AND J. LIM. Speeding up point multiplication on hyperelliptic curves with efficiently-computable endomorphisms. *Advances in Cryptology—EUROCRYPT 2002* (LNCS 2332) [248], 197–208, 2002.

[370] Y. PARK, S. OH, J. L. S. LEE, AND M. SUNG. An improved method of multiplication on certain elliptic curves. *Public Key Cryptography—PKC 2002* (LNCS 2274) [340], 310–322, 2002.

[371] R. PAUL. *SPARC Architecture, Assembly Language Programming, and C*. Prentice Hall, second edition, 2000.

[372] B. PFITZMANN, editor. *Advances in Cryptology—EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Springer-Verlag, 2001.

[373] J. PIEPRZYK, editor. *Advances in Cryptology—ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*. 4th International Conference on the Theory and Application of Cryptology, Wollongong, Australia, November/December 1994, Springer-Verlag, 1995.

[374] R. PINCH. Extending the Wiener attack to RSA-type cryptosystems. *Electronics Letters*, 31:1736–1738, 1995.

[375] L. PINTSOV AND S. VANSTONE. Postal revenue collection in the digital age. *Financial Cryptography—FC 2000* (LNCS 1962) [148], 105–120, 2001.

[376] S. POHLIG AND M. HELLMAN. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.

[377] D. POINTCHEVAL. Chosen-ciphertext security for any one-way cryptosystem. *Public Key Cryptography—PKC 2000* (LNCS 1751) [207], 129–146, 2000.

[378] D. POINTCHEVAL AND J. STERN. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000.

[379] J. POLLARD. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.

[380] ———. Factoring with cubic integers. In Lenstra and Lenstra [280], 4–10.

[381] ———. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13:437–447, 2000.

[382] B. PRENEEL, editor. *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*. International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 2000, Springer-Verlag, 2000.

[383] J. PROOS. Joint sparse forms and generating zero columns when combing. Technical Report CORR 2003-23, Department of Combinatorics and Optimization, University of Waterloo, Canada, 2003.

[384] J. PROOS AND C. ZALKA. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information and Computation*, 3:317–344, 2003.

[385] S. QING, T. OKAMOTO, AND J. ZHOU, editors. *Information and Communications Security 2001*, volume 2229 of *Lecture Notes in Computer Science*. Third Inernational Conference, November 13-16, 2001, Xian, China, Springer-Verlag, 2001.

[386] J. QUISQUATER AND D. SAMYDE. Electromagnetic analysis (EMA): Measures and countermeasures for smart cards. *Smart Card Programming and Security* (LNCS 2140) [21], 200–210, 2001.

[387] J. QUISQUATER AND B. SCHNEIER, editors. *Smart Card Research and Applications*, volume 1820 of *Lecture Notes in Computer Science*. Third International Conference (CARDIS'98), Louvain-la-Neuve, Belgium, September 14-16, 1998, Springer-Verlag, 2000.

[388] J. QUISQUATER AND J. VANDEWALLE, editors. *Advances in Cryptology — EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*. Workshop on the Theory and Application of Cryptographic Techniques, Houthalen, Belgium, April 1989, Springer-Verlag, 1990.

[389] C. RACKOFF AND D. SIMON. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. *Advances in Cryptology—CRYPTO '91* (LNCS 576) [135], 433–444, 1992.

[390] A. REYHANI-MASOLEH AND M. HASAN. Fast normal basis multiplication using general purpose processors. *Selected Areas in Cryptography—SAC 2001* (LNCS 2259) [468], 230–244, 2001.

[391] R. RIVEST, A. SHAMIR, AND L. ADLEMAN. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[392] T. RÖMER AND J. SEIFERT. Information leakage attacks against smart card implementations of the elliptic curve digital signature algorithm. *Smart Card Programming and Security* (LNCS 2140) [21], 211–219, 2001.

[393] B. ROY AND E. OKAMOTO, editors. *Progress in Cryptology—INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*. First International Conference in Cryptology in India, Calcutta, India, December 2000, Springer-Verlag, 2000.

[394] RSA LABORATORIES. PKCS #1 v1.5: RSA Encryption Standard, November 1993.

[395] ———. PKCS #1 v2.1: RSA Cryptography Standard, June 2002.

[396] K. RUBIN AND A. SILVERBERG. The best and worst of supersingular abelian varieties in cryptology. *Advances in Cryptology—CRYPTO 2002* (LNCS 2442) [488], 336–353, 2002.

[397] H. RÜCK. On the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 68:805–806, 1999.

[398] R. RUEPPEL, editor. *Advances in Cryptology—EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*. Workshop on the Theory and Application of Cryptographic Techniques, Balatonfüred, Hungary, May 1992, Springer-Verlag, 1993.

[399] R. SAFAVI-NAINI, editor. *Information Security and Privacy 2003*, volume 2727 of *Lecture Notes in Computer Science*. 8th Australasian Conference, July 9-11, 2003, Wollongong, Australia, Springer-Verlag, 2003.

[400] T. SATOH. The canonical lift of an ordinary elliptic curve over a prime field and its point counting. *Journal of the Ramanujan Mathematical Society*, 15:247–270, 2000.

[401] T. SATOH AND K. ARAKI. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Mathematici Universitatis Sancti Pauli*, 47:81–92, 1998.

[402] T. SATOH, B. SKJERNAA, AND Y. TAGUCHI. Fast computation of canonical lifts of elliptic curves and its application to point counting. *Finite Fields and Their Applications*, 9:89–101, 2003.

[403] E. SAVAŞ AND Ç. KOÇ. The Montgomery inverse—revisited. *IEEE Transactions on Computers*, 49:763–766, 2000.

[404] E. SAVAŞ, A. TENCA, AND Ç. KOÇ. A scalable and unified multiplier architecture for finite fields $GF(p)$ and $GF(2^m)$. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 277–292, 2000.

[405] R. SCHEIDLER, J. BUCHMANN, AND H. WILLIAMS. A key-exchange protocol using real quadratic fields. *Journal of Cryptology*, 7:171–199, 1994.

[406] R. SCHEIDLER, A. STEIN, AND H. WILLIAMS. Key-exchange in real quadratic congruence function fields. *Designs, Codes and Cryptography*, 7:153–174, 1996.

[407] W. SCHINDLER. A timing attack against RSA with the Chinese Remainder Theorem. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 109–124, 2000.

[408] O. SCHIROKAUER. Discrete logarithms and local units. *Philosophical Transactions of the Royal Society of London A*, 345:409–423, 1993.

[409] B. SCHNEIER. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 2nd edition, 1996.

[410] C. SCHNORR. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[411] R. SCHOOF. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44:483–494, 1985.

[412] R. SCHROEPPEL. Automatically solving equations in finite fields. US Patent Application No. 09/834,363, filed 12 April 2001.

[413] ———. Elliptic curves: Twice as fast! Presentation at the CRYPTO 2000 [34] Rump Session, 2000.

[414] R. SCHROEPPEL, C. BEAVER, R. GONZALES, R. MILLER, AND T. DRAELOS. A low-power design for an elliptic curve digital signature chip. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 366–380, 2002.

[415] R. SCHROEPPEL, H. ORMAN, S. O'MALLEY, AND O. SPATSCHECK. Fast key exchange with elliptic curve systems. *Advances in Cryptology—CRYPTO '95* (LNCS 963) [103], 43–56, 1995.

[416] M. SCOTT. Comparison of methods for modular exponentiation on 32-bit Intel 80x86 processors. Informal Draft 11 June 1996. Available from the MIRACL site http://indigo.ie/~mscott/.

[417] SEC 1. Standards for Efficient Cryptography Group: Elliptic Curve Cryptography. Version 1.0, 2000.

[418] SEC 2. Standards for Efficient Cryptography Group: Recommended Elliptic Curve Domain Parameters. Version 1.0, 2000.

[419] R. SEDGEWICK, T. SZYMANSKI, AND A. YAO. The complexity of finding cycles in periodic functions. *SIAM Journal on Computing*, 11:376–390, 1982.

[420] I. SEMAEV. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Mathematics of Computation*, 67:353–356, 1998.

[421] A. SHAMIR. Factoring large numbers with the TWINKLE device. *Cryptographic Hardware and Embedded Systems—CHES '99* (LNCS 1717) [262], 2–12, 1999.

[422] ———. Protecting smart cards from passive power analysis with detached power supplies. *Cryptographic Hardware and Embedded Systems—CHES 2000* (LNCS 1965) [263], 71–77, 2000.

[423] A. SHAMIR AND E. TROMER. Factoring large numbers with the TWIRL device. *Advances in Cryptology—CRYPTO 2003* (LNCS 2729) [55], 1–26, 2003.

[424] P. SHOR. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, 1997.

[425] V. SHOUP. Lower bounds for discrete logarithms and related problems. *Advances in Cryptology—EUROCRYPT '97* (LNCS 1233) [154], 256–266, 1997.

[426] ———. Using hash functions as a hedge against chosen ciphertext attack. *Advances in Cryptology—EUROCRYPT 2000* (LNCS 1807) [382], 275–288, 2000.

[427] ———. OAEP reconsidered. *Journal of Cryptology*, 15:223–249, 2002.

[428] F. SICA, M. CIET, AND J. QUISQUATER. Analysis of the Gallant-Lambert-Vanstone method based on efficient endomorphisms: Elliptic and hyperelliptic curves. *Selected Areas in Cryptography—SAC 2002* (LNCS 2595) [349], 21–36, 2003.

[429] J. SILVERMAN. *The Arithmetic of Elliptic Curves*. Springer-Verlag, 1986.

[430] ———. *Advanced Topics in the Arithmetic of Elliptic Curves*. Springer-Verlag, 1994.

[431] ———. The xedni calculus and the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 20:5–40, 2000.

[432] J. SILVERMAN AND J. SUZUKI. Elliptic curve discrete logarithms and the index calculus. *Advances in Cryptology—ASIACRYPT '98* (LNCS 1514) [352], 110–125, 1998.

[433] J. SILVERMAN AND J. TATE. *Rational Points on Elliptic Curves*. Springer-Verlag, 1992.

[434] R. SILVERMAN AND J. STAPLETON. Contribution to the ANSI X9F1 working group, 1997.

[435] G. SIMMONS, editor. *Contemporary Cryptology: The Science of Information Integrity*. IEEE Press, 1992.

[436] B. SKJERNAA. Satoh's algorithm in characteristic 2. *Mathematics of Computation*, 72:477–487, 2003.

[437] S. SKOROBOGATOV AND R. ANDERSON. Optical fault induction analysis. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 2–12, 2002.

[438] N. SMART. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12:193–196, 1999.

[439] ———. Elliptic curve cryptosystems over small fields of odd characteristic. *Journal of Cryptology*, 12:141–151, 1999.

[440] ———. A comparison of different finite fields for elliptic curve cryptosystems. *Computers and Mathematics with Applications*, 42:91–100, 2001.

[441] ———. The exact security of ECIES in the generic group model. *Cryptography and Coding 2001*, volume 2260 of *Lecture Notes in Computer Science*, 73–84. Springer-Verlag, 2001.

[442] ———. The Hessian form of an elliptic curve. *Cryptographic Hardware and Embedded Systems—CHES 2001* (LNCS 2162) [261], 118–125, 2001.

[443] P. SMITH AND C. SKINNER. A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms. *Advances in Cryptology—ASIACRYPT '94* (LNCS 917) [373], 357–364, 1995.

[444] J. SOLINAS. An improved algorithm for arithmetic on a family of elliptic curves. *Advances in Cryptology—CRYPTO '97* (LNCS 1294) [235], 357–371, 1997.

[445] ———. Generalized Mersenne numbers. Technical Report CORR 99-39, Department of Combinatorics and Optimization, University of Waterloo, Canada, 1999.

[446] ———. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19:195–249, 2000.

[447] ———. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, Department of Combinatorics and Optimization, University of Waterloo, Canada, 2001.

[448] J. SONG, editor. *Information Security and Cryptology '99*, volume 1787 of *Lecture Notes in Computer Science*. Second International Conference, December 9-10, 1999, Seoul, Korea, Springer-Verlag, 2000.

[449] L. SONG AND K. PARHI. Low-energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19:149–166, 1998.

[450] J. SORENSON. An analysis of Lehmer's Euclidean GCD algorithm. *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, 254–258, 1995.

[451] J. STEIN. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1:397–405, 1967.

[452] J. STERN, D. POINTCHEVAL, J. MALONE-LEE, AND N. SMART. Flaws in applying proof methodologies to signature schemes. *Advances in Cryptology—CRYPTO 2002* (LNCS 2442) [488], 93–110, 2002.

[453] D. STINSON, editor. *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*. 13th Annual International Cryptology Conference, Santa Barbara, California, August 1993, Springer-Verlag, 1994.

[454] ———. *Cryptography: Theory and Practice*. CRC Press, 2nd edition, 2002.

[455] D. STINSON AND S. TAVARES, editors. *Selected Areas in Cryptography—SAC 2000*, volume 2012 of *Lecture Notes in Computer Science*. 7th Annual International Workshop, Waterloo, Ontario, Canada, August 14-15, 2000, Springer-Verlag, 2001.

[456] B. SUNAR AND Ç. KOÇ. An efficient optimal normal basis type II multiplier. *IEEE Transactions on Computers*, 50:83–87, 2001.

[457] S. TAVARES AND H. MEIJER, editors. *Selected Areas in Cryptography—SAC '98*, volume 1556 of *Lecture Notes in Computer Science*. 5th Annual International Workshop, Kingston, Ontario, Canada, August 1998, Springer-Verlag, 1999.

[458] E. TESKE. Speeding up Pollard's rho method for computing discrete logarithms. *Algorithmic Number Theory—ANTS-III* (LNCS 1423) [82], 541–554, 1998.

[459] ———. On random walks for Pollard's rho method. *Mathematics of Computation*, 70:809–825, 2001.

[460] E. THOMÉ. Computation of discrete logarithms in $\mathbb{F}_{2^{607}}$. *Advances in Cryptology—ASIACRYPT 2001* (LNCS 2248) [67], 107–124, 2001.

[461] E. TRICHINA AND A. BELLEZZA. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 98–113, 2002.

[462] P. VAN OORSCHOT AND M. WIENER. On Diffie-Hellman key agreement with short exponents. *Advances in Cryptology—EUROCRYPT '96* (LNCS 1070) [306], 332–343, 1996.

[463] ———. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999.

[464] M. VANDERSYPEN, M. STEFFEN, G. BREYTA, C. YANNONI, M. SHERWOOD, AND I. CHUANG. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414:883–887, 2001.

[465] V. VARADHARAJAN AND Y. MU, editors. *Information Security and Privacy 2001*, volume 2119 of *Lecture Notes in Computer Science*. 6th Australasian Conference, July 11-13, 2001, Sydney, Australia, Springer-Verlag, 2001.

[466] S. VAUDENAY. Security flaws induced by CBC padding—applications, to SSL, IPSEC, WTLS... *Advances in Cryptology—EUROCRYPT 2002* (LNCS 2332) [248], 534–545, 2002.

[467] ———. The security of DSA and ECDSA. *Public Key Cryptography—PKC 2003* (LNCS 2567) [116], 309–323, 2003.

[468] S. VAUDENAY AND A. YOUSSEF, editors. *Selected Areas in Cryptography—SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*. 8th Annual International Workshop, Toronto, Ontario, Canada, August 16-17, 2001, Springer-Verlag, 2001.

[469] F. VERCAUTEREN. Computing zeta functions of hyperelliptic curves over finite fields of characteristic 2. *Advances in Cryptology—CRYPTO 2002* (LNCS 2442) [488], 369–384, 2002.

[470] F. VERCAUTEREN, B. PRENEEL, AND J. VANDEWALLE. A memory efficient version of Satoh's algorithm. *Advances in Cryptology—EUROCRYPT 2001* (LNCS 2045) [372], 1–13, 2001.

[471] E. VERHEUL. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. *Advances in Cryptology—EUROCRYPT 2001* (LNCS 2045) [372], 195–210, 2001.

[472] ———. Self-blindable credential certificates from the Weil pairing. *Advances in Cryptology—ASIACRYPT 2001* (LNCS 2248) [67], 533–551, 2001.

[473] J. VIEGA AND G. MCGRAW. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 2001.

[474] L. WASHINGTON. *Elliptic Curves: Number Theory and Cryptography*. CRC Press, 2003.

[475] W. WATERHOUSE. Abelian varieties over finite fields. *Annales Scientifiques de l'École Normale Supérieure, 4^e Série*, 2:521–560, 1969.

[476] D. WEAVER AND T. GERMOND, editors. *The SPARC Architecture Manual, Version 9*. Prentice Hall, 1994.

[477] A. WEIMERSKIRCH, C. PAAR, AND S. CHANG SHANTZ. Elliptic curve cryptography on a Palm OS device. *Information Security and Privacy 2001* (LNCS 2119) [465], 502–513, 2001.

[478] A. WEIMERSKIRCH, D. STEBILA, AND S. CHANG SHANTZ. Generic $GF(2^m)$ arithmetic in software and its application to ECC. *Information Security and Privacy 2003* (LNCS 2727) [399], 79–92, 2003.

[479] A. WENG. Constructing hyperelliptic curves of genus 2 suitable for cryptography. *Mathematics of Computation*, 72:435–458, 2003.

[480] M. WIENER, editor. *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*. 19th Annual International Cryptology Conference, Santa Barbara, California, August 1999, Springer-Verlag, 1999.

[481] ———. The full cost of cryptanalytic attacks. *Journal of Cryptology*, to appear.

[482] M. WIENER AND R. ZUCCHERATO. Faster attacks on elliptic curve cryptosystems. *Selected Areas in Cryptography—SAC '98* (LNCS 1556) [457], 190–200, 1999.

[483] H. WILLIAMS, editor. *Advances in Cryptology—CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986.

[484] C. WITTMANN. Group structure of elliptic curves over finite fields. *Journal of Number Theory*, 88:335–344, 2001.

[485] J. WOLKERSTORFER. Dual-field arithmetic unit for $GF(p)$ and $GF(2^m)$. *Cryptographic Hardware and Embedded Systems—CHES 2002* (LNCS 2523) [238], 500–514, 2002.

[486] A. WOODBURY, D. BAILEY, AND C. PAAR. Elliptic curve cryptography on smart cards without coprocessors. *Smart Card Research and Advanced Applications* [125], 71–92, 2000.

[487] S. YEN, C. LAIH, AND A. LENSTRA. Multi-exponentiation. *IEE Proceedings—Computers and Digital Techniques*, 141:325–326, 1994.

[488] M. YUNG, editor. *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*. 22nd Annual International Cryptology Conference, Santa Barbara, California, August 18-22, 2002, Springer-Verlag, 2002.

[489] Y. ZHENG, editor. *Advances in Cryptology—ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*. 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Springer-Verlag, 2002.

索　引

符号

O-notation(big-O)　大 O 表示法　1.3

o-notation(little-o)　小 o 表示法　1.3

$L_n[\alpha, c]$(subexponential notation)　亚指数表示式 $L_n[\alpha, c]$　1.3

$\mathbb{F}_q$(finite field of order q)　阶 q 的有限域 $\mathbb{F}_q$　2.1

$\mathbb{F}_q^*$(multiplicative group of $\mathbb{F}_q$)　$\mathbb{F}_q$ 的乘法群 $\mathbb{F}_q^*$　2.2

$\mathbb{F}_q$(prime field)　素域 $\mathbb{F}_q$　2.1

$\mathbb{F}_{2^m}$(binary field)　二进制域 $\mathbb{F}_{2^m}$　2.1

$\mathbb{Q}$(the rational number)　有理数域 $\mathbb{Q}$　2.1

$\mathbb{R}$(the real numbers)　实数域 $\mathbb{R}$　2.1

$\mathbb{Z}$(the integers)　整数集 $\mathbb{Z}$　2.4.1

$\oplus$(bitwise exclusive-or)　按位异或 $\oplus$　2.3.1

&(bitwise AND)　按位与&　2.3.1

$\gg i$(right shift by i positions)　右移 i 位　2.3.1

$\ll i$(left shift by i positions)　左移 i 位　2.3.1

∞(point at infinity)　无穷远点 ∞　3.1

$E(L)$(L-rational points on E)　域 L 上椭圆曲线 E 的有理数点 $E(L)$　3.1

$a \| b$(concatenation of strings a, b)　串 a 和 b 的级联　3.3.2

$\#S$(cardinality of a set S)　集合 S 的基数 $\#S$　3.1.3

A

Abelian group　交换群　1.2.3

Access control　访问控制　1.1

Additive group　加法群　1.2.3

Admissible change of variables　变量的相容性变换　3.1.1

Advanced Encryption Standard, see AES　高级加密标准，见 AES

Adversarial model　敌手模型　1.1

AES　AES　1.1

　Large　大规格～　1.3

　Medium　中规模～　1.3

　Small　小规格～　1.3

Affine coordinates　仿射坐标　3.1.2

Affine point　仿射点　3.2.1

AGM algorithm　AGM 算法　4.3, 4.7

Algorithm　算法

　exponential-time　指数时间～　1.3

　fully-exponential-time　全指数时间～　1.3

　polynomial-time　多项式时间～　1.3

　running time　～运行时间　1.3

　subexponential-time　亚指数时间～　1.3

Alignment　队列　5.1.4

Almost cyclic　殆循环　3.1.5

Almost inverse algorithm　殆逆算法　2.3.6, 5.1.5

Almost prime　殆素数　3.4, 4.2.1

American National Standards Institute, see ANSI　美国国家标准研究所，见 ANSI

Anomalous binary curve, see Koblitz cure　异常二进制曲线，见 Koblitz 曲线

Anonymity　匿名性　1.1

ANSI　ANSI　附录 B

　X9.62　4.2.2, 4.4.1, A.1, A.1, 附录 B

　X9.63　4.5.1, 4.6.1, 4.6.2, A.1, A.1, 附录 B

ASIC　ASIC　5.2

B

Barrett reduction　Barrett 约减　2.2.4, 2.5, 5.1.5

Base point　基点　4.2

Big-O notation　大 O 表示法　1.3

Binary field　二进制域　2.1

addition ~加法 2.3, 5.2.2
arithmetic with MMX 使用 MMX 的~算术 5.1.3
division ~除法 2.3.6, 5.1.5
inversion ~求逆 2.3.6, 5.1.5, 5.2.2
Karatsuba-Ofman multiplication ~Karatsuba- Ofman 乘法 2.3.3
multiplication ~乘法 2.3.2, 5.1.5, 5.2.2
polynomial multiplication ~多项式乘法 2.3.2
polynomial squaring ~多项式平方 2.3.4
reduction ~约减 2.3.5
squaring ~平方 5.2.2
timings ~时间比较 5.1.5
Binary inversion algorithm 二进制逆算法
for binary fields 二进制域的~ 2.3.6, 5.1.5
for prime fields 素域的~ 2.2.5
Birthday paradox 生日悖论 4.1.2
Bit-Serial multiplier 位串行乘法器 5.2.2
Bleichenbacher's attack Bleichenbacher 攻击 5.4
Branch misprediction 分支预测失误 5.1.4

C

Carry bit 进位 2.2.1
Certicom ECDLP challenge Certicom ECDLP 挑战 1.5
Characteristic 特征 2.1
Characteristic-two finite field 特征为 2 的有限域 2.1
Chudnovsky coordinates Chudnovsky 坐标 3.2.2, 3.8
CM method 复乘法（CM）方法 4.2.3
co-NP 4.1
Cofactor 余因子 3.4, 4.2
Collision 碰撞 4.1.2
Comb method Comb 方法
for point multiplication 用于点乘的~ 3.3.2
for polynomial multiplication 用于多项式乘的~ 2.3.3
Confidentiality 秘密性 1.1
Coordinates 坐标
affine 仿射~ 3.1.2
Chudnovsky Chudnovsky~ 3.2.2, 3.8
Jacobian 雅可比~ 3.2.1, 3.2.2, 3.2.3
LD LD~ 3.2.3, 3.8
projective 投影~ 3.2.1
Cost-equivalent key sizes 成本等价密钥规模 1.4
Cramer-Shoup public-key encryption Cramer-Shoup 公钥加密方案 4.7
Cryptographic Research and Evaluation Committee, see CRYPTREC 密码研究与评估委员会，见 CRYPTREC
CRYPTREC CRYPTREC 4.5.2, 附录 B
Cyclic group 循环群 1.2.3
generator ~生成器 1.2.3
Cyclic subgroup 循环子群 1.2.3

D

Data encapsulation mechanism 数据封装机制 4.5.2
Data Encryption Standard 数据加密标准 1.1
Data integrity 数据完整性 1.1
Data origin authentication 数据源认证 1.1
DES 1.1
Differential power analysis, see DPA 差分能量分析，见 DPA
Differential trace 差分轨迹 5.3.1
Diffie-Hellman problem Diffie-Hellman 问题 1.2.2
Digit-serial multiplier 位串乘法器 5.2.2
Digital Signature Algorithm(DSA) 数字签名算法（DSA）1.2.2
Digital Signature Standard 数据签名标准 1.2.2
Discrete logarithm systems 离散对数体制 1.2.2
basic encryption scheme ~基本加密方案 1.2.2
domain parameter generation ~参数组生成 1.2.2
key pair generation ~密钥对生成 1.2.2
signature scheme ~签名方案 1.2.2
Discriminant 判别式 3.1
Distinguished point 特殊点 4.1.2
Division in binary fields 二进制域的除法 2.3.6, 5.1.5
Domain parameters 参数组 4.2, A.1, A.2
generation ~生成 4.2.1
validation ~确认 4.2.2
DPA DPA 5.3.1, 5.4
DSA DSA 1.2.2

E

Early-abort strategy 早期中止策略 4.2.1, 4.3
EC-KCDSA EC-KCDSA 4.4.2, 4.7

ECDLP, see elliptic curve discrete logarithm problem ECDLP，见椭圆曲线离散对数问题
ECDSA ECDSA 4.4.1, 4.7
ECIES ECIES 4.5.1, 4.7
ECMQV ECMQV 4.6.2, 4.7
Efficient algorithm 有效算法 1.3
Electromagnetic analysis attacks 电磁分析攻击 5.3.2, 5.4
ElGamal encryption ElGamal 加密 1.2.2
Elliptic curve 椭圆曲线 1.2.2
admissible change of variables ~变量的相容性变换 3.1.1
affine coordinates ~仿射坐标 3.1.2
affine point ~仿射点 3.2.1
Chudnovsky coordinates Chudnovsky 坐标 3.2.2, 3.8
definition 的定义 3.1
discriminant 判别式 3.1
double of points 倍点 3.1.2
endomorphism 自同态 3.5
group law 群的运算法则 3.1.2
group structure 群的结构 3.1.4
Hessian form Hessian 齐式 3.8, 5.4
isogenous 同源 4.7
isomorphic 同构 3.1.1
isomorphism classes 同构类 3.1.5
Jacobi form 雅可比齐式 3.8, 5.4
Jacobi model 雅可比型 3.8
Jacobian coordinates 雅可比坐标 3.2.1, 3.2.2, 3.2.3
LD coordinates LD 坐标 3.2.3, 3.8
non-supersiingular 非超奇异 3.1.1, 3.1.4
order 阶 3.1.3
point 点 1.2.3
point at infinity 无穷远点 1.2.3, 3.1
projective point 投影点 3.2.1
selecting verifiably at random 可验证随机选择 3.1
sum of points 点的和 4.2.1
supersingular 超奇异 3.1.2
trace 迹 3.1.2, 3.1.4
underlying field 基础域 3.1
Weierstrass equation Weierstrass 方程 3.1
Elliptic curve decision Diffie-Hellman problem 椭圆曲线判定性 Diffie-Hellman 问题 4.2
Elliptic curve Diffie-Hellman problem 椭圆曲线 Diffie-Hellman 问题 4.1.5, 4.7
Elliptic curve discrete logarithm problem 椭圆曲线离散对数问题 1.2.3, 4.1
GHS attack GHS 攻击 4.1.4, 4.7
index-calculus attack 索引计算攻击 4.1.3
kangaroo algorithm 袋鼠算法 4.7
Lambda method Lambda 方法 4.7
parallelized Pollard's rho attack 并行 Pollard's rho 攻击 4.1.2
Pohlig-Hellman attack Pohlig-Hellman 攻击 4.1.1
Pollard's rho attack Pollard's rho 攻击 4.1.2, 4.7
prime-field-anomalous curves 素域异常曲线 4.1.4, 4.7
Tate pairing attack Tate 配对攻击 4.1.4, 4.7
Weil descent attack Weil 下降攻击 4.1.4, 4.7
Weil pairing attack Weil 配对攻击 4.1.4, 4.7
xedni calculus xedni 计算 4.7
Elliptic curve systems 椭圆曲线体制 1.2.3
basic ELGamal encryption 基本 ElGamal 加密 4.4.2
EC-KCDSA EC-KCDSA 4.4.1
ECDSA ECDSA 4.4.1
ECIES ECIES 4.5.1
ECMQV ECMQV 4.6.2
key pair generation 密钥对生成 1.2.3
PSEC PSEC 4.5.2
station-to-station 站对站 4.6.1
Embedding degree 嵌入度 4.1.4
Endomorphism 自同态
definition of ~的定义 3.5
efficiently computable 有效可计算的~ 3.5, 3.8
point multiplication ~点乘 3.6
Frobenius Frobenius~ 3.5
ring ~环 3.5
Entity authentication 实体认证 1.1
Error message analysis 错误消息分析 5.3.3
Explicit key authentication 显式密钥认证 4.6.1
Exponent array 代表矩阵 3.3.2, 3.3.3
Exponential-time algorithm 指数时间算法 1.3

Extended Euclidean algorithm 扩展的欧几里得算法
for integers 整数的～ 2.2.5
for polynomials 多项式的～ 2.3.6, 5.1.5
Extension field 扩域 2.1

F

Factor base 因数基 4.1.3
Fault analysis 错误分析 5.3.4, 5.4
Federal Information Processing Standards, see FIPS 联邦信息处理标准，见 FIPS
Field 域 2.1
Finite field 有限域 1.2.3, 2.1
binary 二进制～ 2.1
characteristic ～的特征 2.1
extension 扩域 2.1
isomorphic ～的同构 2.1
order ～的阶 2.1
prime 素～ 2.1
primitive element 本原元 2.4.2
subfield, see also binary field, prime field, optimal extension field 子域，参见二进制域、素域、最佳扩域 2.1
FIPS, see NIST FIPS，见 NIST
Floating-point arithmetic 浮点算术 5.1.2, 5.2
Floyd's cycle-finding algorithm Floyd 循环查找算法 4.1.2
Forward secrecy 前向保密 4.6.1
FPGA FPGA 5.2
Frobenius map Frobenius 映射 2.4.3, 3.4, 3.5
Fully-exponential-time algorithm 全指数时间算法 1.3

G

Gate 门 5.2
Gate array 门阵列 5.2
Gaussian normal basis 高斯正则基 2.5, A.2.3
Generator 生成器 1.2.3
Generic group 一般群 4.1
GH GH 1.5
GHS attack GHS 攻击 4.1.4, 4.7
GMR-secure GMR 安全 4.4
GNU MP(gmp) GNU MP 5.1.2, 5.1.4, C.2
Greatest common divisor 最大公因子
of integers 整数的～ 2.2.5
of polynomials 多项式的～ 2.3.6
group 群 1.2.3
Generic group 一般群 4.1
Group law 群的运算法则 3.1.2

H

Half-trace function 半迹函数 3.6.2
Hasse interval Hasse 区间 3.1.3
Hasse's Theorem Hasse 定理 3.1.3
Hessian form Hessian 齐式 3.8, 5.4
HMAC HMAC 1.1
Hyperelliptic curve 超椭圆曲线 1.5, 3.8, 4.1.3, 4.7
Hyperelliptic curve discrete logarithm problem 超椭圆曲线离散对数问题 4.1.4

I

IEEE IEEE 附录 B
1363-2000 1363-2001 4.4.1, 4.6.2, 附录 B
P1363a P1364a 4.5.1, 附录 B
IKE IKE 4.7
Implicit key authentication 隐式密钥认证 4.6.1
Implicit signature 隐式签名 4.6.2
Index-calculus attack 索引计算攻击 4.1.3
Institute of Electrical and Electronics Engineers, see IEEE 电气与电子工程师协会，见 IEEE
Integer 整数
arithmetic with floating-point 浮点～算术 5.1.2, 5.2
Karatsuba-Ofman multiplication Karatsuba-Ofman 乘 2.2.2
multiplication 乘 2.2.2, 5.1.1
reduction 约减 2.2.4
squaring 平方 2.2.3
Integer factorization problem 整数因子分解问题 1.2.1
Interleaving 交错 3.3.3
International Organization for Standardization, see ISO/IEC 国际标准化组织，见 ISO/IEC
Invalid-curve attack 无效曲线攻击 4.3, 4.7
Inversion 求逆

in binary fields 二进制域～ 2.3.6, 5.1.5, 5.2.2
in optimal extension fields 最佳扩域～ 2.4.3
in prime fields 素域～ 2.2.5
Irreducible polynomial 既约多项式 A.1
ISO/IEC ISO/IEC 附录 B
15946-1 15946-1 附录 B
15946-2 15946-2 4.4.1, 4.4.2, 附录 B
15946-3 15946-3 4.5.1, 4.6.2, 附录 B
15946-4 15946-4 附录 B
18033-2 18033-3 附录 B
Isogenous elliptic curves 同源椭圆曲线 4.7
Isomorphic 同构
elliptic curves 椭圆曲线～ 3.1.1, 3.1.5
fields 域～ 2.1

J

Jacobi form 雅可比齐式 3.8, 5.4
Jacobi model 雅可比型 3.8
Jacobian coordinates 雅可比坐标 3.2.1, 3.2.2, 3.2.3
Joint sparse form 联合稀疏型 3.3.3, 3.8

K

Kangaroo algorithm 袋鼠算法 4.7
Karatsuba-Ofman multiplication Karatsuba-Ofman 乘
for integers 整数的～ 2.2.2, 5.1.5
for polynomials 多项式的～ 2.3.3
Kedlaya's algorithm Kedlaya 算法 4.7
Key agreement protocol, see also key establishment 密钥协商协议，参见密钥建立 4.6
Key confirmation 密钥确认 4.6.1
Key derivation function 密钥生成函数 4.3, 4.5.1, 4.5.2
Key distribution problem 密钥分发问题 1.1
Key encapsulation mechanism 密钥封装机制 4.5.2
Key establishment 密钥建立 4.6
ECMQV ECMQV 4.6.2, 4.7
IKE IKE 4.7
OAKLEY OAKLEY 4.7
security 安全性 4.6
SKEME SKEME 4.7
station-to-station 站对站 4.6.1, 4.7
Key management problem 密钥管理问题 1.1
Key pair 密钥对 4.3
generation ～生成 1.2.3, 4.3
validation ～确认 4.3, 4.7
Key transport protocol, see also key establishment 密钥传输协议，参见密钥建立 4.6
Key-compromise impersonation resilience 密钥泄漏后冒充的防护 4.6.1
Koblitz curve Koblitz 曲线 4.1.2, A.2.2
almost-prime group order 殆素群阶 3.4
TNAF TNAF 3.4.1
TNAF method TNAF 方法 3.4.2
window TNAF method 窗口 TNAF 方法 3.5

L

Lambda method Lambda 方法 4.7
Latency 延迟 5.1.1
LD coordinates LD 坐标 3.2.3, 3.8
Lehmer's gcd algorithm Lehmer 最大公因子算法 2.5
Lim-Lee exponentiation method Lim-Lee 求幂方法 3.3.2
Line at infinity 无穷远的线 3.2.1
Little-o notation 小 o 表示 1.3
LSB multiplier 低位优先乘法器 5.2.2
LUC LUC 1.5

M

Möbius function Mobius 函数 A.1
MAC MAC 1.1
Malleability 扩展性 4.5.1
Modulus 模 2.1
Monic polynomial 首一多项式 A.1
Montgomery Montgomery
inversion ～逆 2.2.5, 2.5, .4
multiplication ～乘 2.2.4
point multiplication ～点乘 3.3.1, 5.4
reduction ～约减 2.2.4, 2.5
Mordell-Weil Theorem Mordell-Weil 定理 4.1.3
MSB multiplier 高位优先乘法器 5.2.2
Multiplexor 多路复用器 5.2.1
Multiplicative group 乘法群 1.2.3

N

NAF NAF 3.3.1
National Institute of Standards and Technology, see NIST 美国标准与技术研究所，见 NIST
NESSIE NESSIE 4.5.2，附录 B
New European Schemes for Signatures, Integrity and Encryption, see NESSIE 最新欧洲签名、完整性和加密方案，见 NESSIE
NIST NIST 附录 B
FIPS 180-2 FIPS 180-3 附录 B
FIPS 186 FIPS 187 1.2.2
FIPS 186-2 FIPS 186-3 4.4.1, A.1, A.2, 附录 B
FIPS 197 FIPS 197 附录 B
FIPS 198 FIPS 198 附录 B
FIPS 46 FIPS 46 附录 B
FIPS 44 FIPS 44 5.1.5
reduction polynomial 约减多项式 2.3.5, 5.1.5
Non-adjacent form(NAF) 非相邻表示型（NAF） 3.3.1
Non-repudiation 不可否认 1.1
Non-supersingular 非超奇异 3.1.1, 3.1.4
Normal basis 正则基 2.5, 3.6.2, 5.4, A.2.3
NP NP 4.1
NP-hard NP 困难的 4.1
Number Field Sieve 数域筛法 1.3

O

OAKLEY OAKLEY 4.7
OEF, see optimal extension field OEF，见最佳扩域
OpenSSL, see also SSL OpeSSL，参见 SSL 2.3.4, 5.4, C.2
Optical fault induction attack 光学故障感应攻击 5.4
Optimal extension field 最佳扩域
addition ~加法 2.4.1
inversion ~求逆 2.4.3
multiplication ~乘法 2.4.1
reduction ~约减 2.4.1
subtraction ~减法 2.4.1
timings ~时间比较 5.1.5
Type 最佳扩域的型 2.4
Optimal normal basis 最佳正则基 2.5
Order 阶
of a field element 域元素的~ 2.2
of a finite field 有限域的~ 2.1
of a group 群的~ 1.2.3
of a group element 群元素的~ 1.2.3
of an elliptic curve 椭圆曲线的~ 3.1.3

P

Parallel processing 并行处理 5.2.1
Parallelized Pollard's rho attack 并行 Pollard's rho 攻击 4.1.2
Pentanomial 五项式 2.3.5, 3.6.1, A.1
Pipelining 流水线操作 5.2.1
Pohig-Hellman attack Pohig-Hellman 攻击 4.1.1
Point 点 1.2.3
double 倍~ 3.1.2
sum ~的和 3.1.2
Point at infinity 无穷远点 1.2.3, 3.1
Point counting algorithms 点计数算法 4.2.3, 4.7
Point halving 点折半 3.6, 3.8
Have-and-add 折半加 3.6.3
Point multiplication 点乘 3.3
binary NAF method 二进制 NAF 方法 3.3.1
comparisons 点乘比较 3.7
fixed-base comb method 固定基 comb 方法 3.3.2
fixed-base NAF windowing method 固定基 NAF 窗口方法 3.3.2
fixed-base windowing method 固定基窗口方法 3.3.2
halve-and-add 点折半 3.6.3
interleaving 交错 3.3.2
left-to-right binary method 从左向右的二进制方法 3.3.1
Lim-Lee method Lim-Lee 方法 3.3.2
right-to-left binary method 从右向左的二进制方法 3.3.1
sliding window method 滑动窗口方法 3.3.3
timings 时间比较 3.7
TNAF method TNAF 方法 3.4.2
window NAF method 窗口 NAF 方法 3.3.1
window TNAF method 窗口 TNAF 方法 3.5
with efficiently computable endomorphisms 具有有效可计算自同态的点乘 3.6
Pollard's rho attack Pollard's rho 攻击 1.3, 4.1.2, 4.7
Polynomial 多项式

Karatsuba-Ofman multiplication Karatsuba- Ofman 乘法 2.3.3
multiplication 乘法 2.3.2
reduction 约减 2.3.5
squaring 平方 2.3.4
Polynomial basis 多项式基 2.1
Polynomial security 多项式安全 4.7
Polynomial-time algorithm 多项式时间算法 1.3
Power analysis 能量分析 5.3.1
DPA DPA 5.3.1, 5.4
SPA SPA 5.3.1, 5.4
Power trace 能量轨迹 5.3.1
Prime field 素域 2.1
addition 加法 2.2.1
arithmetic with SIMD 用 SIMD 进行算术运算 5.1.3, 5.2, 5.4
integer multiplication 整数乘法 2.2.2
integer squaring 整数平方 2.2.3
inversion 求逆 2.2.5
Karatsuba-Ofman multiplication Karatsuba- Ofman 乘法 2.2.2, 5.1.5
reduction 约减 2.2.4
subtraction 减法 2.2.1
timings 时间比较 5.1.5
Prime-field-anomalous curve 素域异常椭圆曲线 4.1.4, 4.7
Primitive element 本原元 2.4.1
Program optimizations 程序优化
assembly coding 汇编代码 5.1.4
duplicated code 重复代码 5.1.4
loop unrolling 循环展开 5.1.4
Projective coordinates, see coordinates 投影坐标，见坐标
Projective point 投影点 3.2.1
PSEC PSEC 4.5.2
Public key validation 公钥确认 4.3, 4.7
Public-key cryptography 公钥密码 1.1
public-key encryption 公钥加密 4.5
Cramer-Shoup Cramer-Shoup 方案 4.7
ECIES ECIES 4.5.1, 4.7
malleability 可扩展性 4.5.1
polynomial security 多项式安全 4.7
PSEC PSEC 4.5.2
security 安全性 4.5
semantic security 语义安全性 4.7
Public-key infrastructure 公钥基础设施 1.1

Q

Quadratic number field 二次数域 1.5, 4.1.3
Quantum computer 量子计算机 4.7
Qubit Qubit 4.7

R

Rational points 有理数点 3.1
RC4 RC4 1.1
Reduction 约减
Barrett Barrett 约减 2.2.4, 2.5, 5.1.5
Montgomery Montgmery 方法 2.2.4, 2.5
Polynomial 多项式约减 2.1
RSA RSA 1.2.1
basic encryption scheme 基本加密方案 1.2.1
basic signature scheme 基本签名方案 1.2.1
FDH FDH 5.3.4
key pair generation 密钥对生成 1.2.1
OAEP OAFP 5.3.3, 5.4
PSS PSS 5.3.4
Running time 运行时间 1.3

S

Satoh's algorithm Satoh 算法 4.3, 4.7
Scalar multiplication, see point multiplication 标量乘，见点乘
Schoof's algorithm Schoof 算法 4.2.3, 4.7
SEA algorithm SEA 算法 4.2.3, 4.7
SECG SECG 附录 B
Security level 安全程度 1.3
Semantic security 语义安全性 4.7
Session key 会话密钥 4.6
SHA-1 SHA-1 4.2.1
Shamir's trick Shamir 窍门 3.3.3
Side-channel attack 旁门攻击 5.3
electromagnetic analysis 电磁分析 5.3.2

error message analysis 错误消息分析 5.3.3
fault analysis 故障分析 5.3.4
optical fault inductin 光学故障 5.4
power analysis 能量分析 5.3.1
timing 时间分析 5.3.5
Signature scheme 签名方案
EC-KCDSA EC-KCDSA 4.4.2, 4.7
ECDSA ECDSA 4.4.1, 4.7
Security 安全性 4.4
Signed digit representation 带符号数字表示 3.3.1
SIMD SIMD 5.1.3, 5.2, 5.4
Simple power analysis, see SPA 简单能量分析，见 SPA
Simultaneous inversion 同时求逆 2.2.6
Single-instruction multiple-data, see SIMD 单指令多数据，见 SIMD
SKEME SKEME 4.7
SKIPJACK SKIPJACK 1.3
Small subgroup attack 小子群攻击 4.3, 4.7
SPA SPA 5.3.1, 5.4
SSL, see also OpenSSL SSL，参见 OpenSSL 4.3, 5.2.1, 5.3.5, 5.4
SST algorithm SST 算法 4.3, 4.7
Standards 标准 附录 B
ANSI ANSI 附录 B
CRYPTREC CRYPTREC 4.5.2, 附录 B
FIPS FIPS 附录 B
IEEE IEEE 附录 B
ISO/IEC ISO/IEC 附录 B
NESSIE NESSIE 4.5.2, 附录 B
NIST NIST 附录 B
SECG SECG 附录 B
Standards for Efficient Cryptography Group(SECG) 密码标准化组织（SECG） 附录 B
Station-to-station protocol 站对站协议 4.6.1, 4.7
STS, see station-to-station protocol STS
Subexponential-time algorithm 亚指数时间算法 1.3
Subfield 子域 2.1
Superelliptic curve 超椭圆曲线 1.5
Supersingular 超奇异 3.1.2, 3.1.4
Symmetric-key cryptography 对称密钥密码 1.1

T

Tage pairing attack Tate 配对攻击 4.1.4, 4.7
Throughput 吞吐量 5.1.1
Timing attack 时间分析攻击 5.4
TNAF TNAF 3.4.1
Trace 迹
function ~函数 3.6.1
of an elliptic curve 椭圆曲线的~ 3.1.3
Trinomial 三项式 2.3.5, 3.6.1, A.1
Triple-DES 三重 DES 1.3

U

Underlying field 基础域 3.1
Unknown key-Share resilience 未知密钥共享的防护 4.6.1

V

VLSI VLSI 5.2

W

Weierstrass equation Weierstrass 方程 3.1
Weil Weil
descent attack ~下降攻击 4.1.4, 4.7
pairing attack ~配对攻击 4.1.4, 4.7
Width-w 宽度 w
NAF NAF 3.3.1
TNAF TNAF 3.4.2

X

Xedni calculus Xedni 计算 4.7
XTR XTR 1.5